



Python

Here're some Python basics you need to know

About language:

▼ What is Python used for?

- web-development (server-side)
- software development
- mathematics
- system scripting

▼ What can Python do?

- Python can be used on a server to web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

▼ Why Python?

- Python works on different platforms, OS (Windows, Mac, Linux, Raspberry, Pi, etc.).
- Python has a simple syntax to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

▼ Gear: IDE, frameworks

- IDE (Integrated Development Environment): Thonny, PyCharm, Netbeans, Eclipse.
- Frameworks: Django.

▼ Python compared to other languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation (табуляция), using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

▼ Python and Functional Programming

Python is a multi-paradigm language. It supports imperative, object oriented, as well as functional programming approach.

Functional programming paradigm recommends dividing the programming logic into set of functions that only take input and produce output not that affects the output produced for a given input.

Features of Python which help in writing functional-style programs: lambda function, recursion, iterator, generator, list comprehension, the itertools module, map(), filter(), reduce().

Python and OOP: <https://www.knowledgehut.com/tutorials/python-tutorial/python-object-oriented-programming>

Variables

Variables are containers for strong data values; they are links to data that is stored in a computer's operative memory.

A variable is created (and its type is being declared) as soon as one assigns a value to it, and can be changed anytime.

▼ Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)
y = int(3)
z = float(3)
print(x)
print(y)
print(z)
```

→

3

3

3.0

▼ Get Type

▼ You can get the type of a variable by using type() function

```
x = str(3)
y = int(3)
z = float(3)
print(type(x))
print(type(y))
print(type(z))
```

→

<class 'str'>

<class 'int'>

<class 'float'>

▼ Names

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- Multi-names can be written in: camel case (myVariableName), Pascal case (MyVariableName), snake case (my_variable_name).

▼ Assign Multiple Values

▼ Many values to multiple variables

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

→

Orange

Banana

Cherry

▼ One value to multiple variables

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

→

Orange

Orange

Orange

▼ Unpack a collection

```
fruits = ["Apple", "Pear", "Kiwi"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

→

Apple

Pear

Kiwi

▼ Output Variables

Usually - with print().

▼ Also can place several variables in one line:

```
fruits = ["Apple ", "Pear ", "Kiwi"]
x, y, z = fruits
print(x+y+z)
```

→

Apple Pear Kiwi

▼ Global Variables

▼ These are variables that are created outside of a function. They can be used everywhere, by everyone, both outside functions and inside.

```
x = "Python"

def pyfunc_my():
    print(x+" is awesome!")

pyfunc_my()
```

→

Python is awesome!

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

- ▼ To create a global variable inside a function, you can use the `global` keyword.

```
def pyfunc_my():
    global x
    x = "Python"
    print(x+" is awesome!")

pyfunc_my()

print(x+" is not that hard.")
```

→

Python is awesome!

Python is not that hard.

- ▼ Also, use the `global` keyword if you want to change a global variable inside a function.

```
x = "Java"

print(x)

def pyfunc_my():
    global x
    x = "Python"
    print(x+" is awesome!")

pyfunc_my()

print(x+" is not that hard.")
```

→

Java

Python is awesome!

Python is not that hard.

Data Types:

Integers

These are simply whole numbers. Shortened as ‘int’. E.g.: 8 and -8. (also known as “numeric type”)

When a=25 the value is called “integer literal”. Function int(“25”) - converts str to int (but it’s not integer literal).

To divide a long number, like 1000000, one can use underscores(): 1_000_000.

Floating-Point Numbers

These are numbers with a decimal point. Shortened as ‘float’. E.g.: 8.0 and -0.9 and 1.3. Max = 2×10^{400} (2e400 → inf). (also known as “numeric type”)

When a = 2.5 - floating point literal, but float(“2.5”) (converts str to float) - no.

- ▼ 1000000.0 is 1_000_000.0 is 1e6 (E[xponential] notation, here equals 1×10^6).

```
>>> 20000000000000000000.0
2e+17 #+ means that the exponent 17 is positive
```

```
>>> 1e-4
0.0001 #raised in power -4
```

Complex Numbers

Have <real part>+<imaginary part>. E.g.: 2+3j will print (2+3j) (also known as “numeric type”).

▼ Operations with numbers

“+” - addition (e.g. $2 + 3 = 5$ and $1.0 + 3 = 4.0$)

“-” - subtraction (e.g. $3 - 2 = 1$ or $4.0 - 2 = 2.0$ or just -3)

“*” - multiplication (e.g. $2 * 2 = 4$ or $2.0 * 2 = 4.0$)

“/” - division (e.g. $\text{int}(4/2) = 2$ or $4/2 = 2.0$ or $9.0/3 = 3.0$ or $\text{int}(5.0/2) = 2$)

“//” - integer division or floor division (e.g. $5//2 = 2$ or $-3//2 = -2$, b.c. it round down the number)

“**” - power exponent (e.g. $2 ** 3 = 8$ or $9 ** 0.5 = 3.0$ or $2 ** -1 = 0.5$)

▼ “%” - modulus operator (e.g. 5%3 = 2 or 6%3 = 0)

Things get a little tricky when you use the `%` operator with negative numbers:

```
>>>
```

```
>>> 5 % -3  
-1
```

```
>>> -5 % 3  
1
```

```
>>> -5 % -3  
-2
```

Although potentially shocking at first glance, these results are the product of a well-defined behavior in Python. To calculate the remainder `r` of dividing a number `x` by a number `y`, Python uses the equation `r = x - (y * (x // y))`.

For example, to find `5 % -3`, Python first finds `(5 // -3)`. Since `5 / -3` is about `-1.67`, that means `5 // -3` is `-2`. Now Python multiplies that by `-3` to get `6`. Finally, Python subtracts `6` from `5` to get `-1`.

▼ Python Assignment Operators

- “`+=`” — example: `x += 3` — same as: `x = x + 3`
- “`-=`” — example: `x -= 3` — same as: `x = x - 3`
- “`*=`” — example: `x *= 3` — same as: `x = x * 3`
- “`/=`” — example: `x /= 3` — same as: `x = x / 3`
- “`%=`” — example: `x %= 3` — same as: `x = x % 3`
- “`//=`” — example: `x //= 3` — same as: `x = x // 3`
- “`**=`” — example: `x **= 3` — same as: `x = x ** 3`
- “`&=`” — example: `x &= 3` — same as: `x = x & 3`
- “`|=`” — example: `x |= 3` — same as: `x = x | 3`
- “`^=`” — example: `x ^= 3` — same as: `x = x ^ 3`
- “`<<=`” — example: `x <= 3` — same as: `x = x << 3`

- “`>>=`” — example: `x >>= 3` — same as: `x = x >> 3`

▼ Other Operator Groups

- Comparison: `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Logical: and, or, not.
- Identity: `is`, `is not`.
- Membership: `in`, `not in`.

▼ Bitwise

Bitwise

<u>Aa</u> Operator	≡ Name	≡ Description	📅 Date
<u>May spa</u>	may spa	Summer prep: lilac, lilly of the vally, cherry blossom vibes	@May 21, 2023
<u>March spa</u>	march spa	Women's day prep: only flowers, bubbles and bright colors	@March 5, 2023
<u>December</u>	December spa	New Year prep: vanilla, gingerbread cookies, choco and shiny winter shimmer	@December 25, 2022
<u>June spa</u>	june spa	Birthday prep: sweet cake, bright sparkles and bubbles	@June 4, 2023
<u>October</u>	October spa	Halloween prep: pumpkin'n'autumn vibes, cozy and funny	@October 30, 2022
<u>February</u>	February spa	Valentine's day prep: wine, movies, strawberries in chocolate mood	@February 12, 2023
<u>September</u>	september spa	Autumn vibes: apple pie, cinnamon roll, pancakes with maple syrup	@September 23, 2023

▼ Integer and Floating Point Methods

- ▼ `num.is_integer()` - returns True if int

```
num = 5.6
print(num.is_integer())
num1 = 5.0
print(num1.is_integer())
```

→

False

True

▼ f-string for numbers

```
>>> n = 7.125
>>> f"The value of n is {n}"
'The value of n is 7.125'
```

Those curly braces support a simple [formatting language](#) that you can use to alter the appearance of the value in the final formatted string.

```
>>> n = 7.125
>>> f"The value of n is {n:.2f}"
'The value of n is 7.12'
```

```
>>> n = 1234567890
>>> f"The value of n is {n:,}"
'The value of n is 1,234,567,890'
```

The specifier `, .2f` is useful for displaying currency values:>>>

```
>>> balance = 2000.0
>>> spent = 256.35
>>> remaining = balance - spent

>>> f"After spending ${spent:.2f}, I was left with ${remaining:.2f}"
'After spending $256.35, I was left with $1,743.65'
```

Another useful option is `%` , which is used to display percentages.

```
>>> ratio = 0.9
>>> f"Over {ratio:.1%} of Pythonistas say 'Real Python rocks!''"
"Over 90.0% of Pythonistas say 'Real Python rocks!'"

>>> # Display percentage with 2 decimal places
```

```
>>> f"Over {ratio:.2%} of Pythonistas say 'Real Python rocks!'"  
"Over 90.00% of Pythonistas say 'Real Python rocks!'"
```

Strings

These are sequences of character data. Shortened as ‘str’. E.g.: “Hey there!” or “123” or ‘’. (also known as “text type”)

If needt to include ‘ or “ in a str - “st’r”. Other - ‘st”r’.

▼ Escape Sequences ()

\' - leaves ‘ in ‘str’

\\" - leaves “ in “str”

\<new line> - newline is ignored, terminates input line

\\\ - leaves \ (backslash)

\a - ASCII Bell (**BEL**) character - nothing will be printed

\b - ASCII Backspace (**BS**) character - all in one line

\f - ASCII Formfeed (**FF**) character - new line saving all spaces

\n - new line saving all spaces

\t - tab space (horizontal)

\r - return to the begginigs of a str

\xxxx - Unicode character with 16-bit hex value **xxxx**

\Uxxxxxxxxx - Unicode character with 32-bit hex value **xxxxxxxx**

\v - ASCII Vertical Tab (**VT**) character - tab space (vertical)

\ooo - Character with octal value **ooo**

\xhh - Character with hex value **hh**

\N{name} - Character named ‘name’ in the Unicode database

Raw String - prefix ‘r’ ot ‘R’ before str - leaves all the escapes. E.g.: r’Hel\nlo’ → Hel\nlo.

Tripple-quoted str - tripple quotes enable ‘ and “ and newlines be leaft, all the escapes stay.

E.g.: “”” I’m so in love with \n\t YOU!””” →

I’m so in love with

YOU!

▼ Slicing strings

```
x = 'A fat cat sat on a mat'  
print(x[2:21:2])  
print(x[::-1])
```

→

ftctsto a
tam a no tas tac taf A

▼ String methods

▼ str.capitalize() - converts the first character to upper case

```
b = 'hey, honey!'  
print(b)  
print(b.capitalize())
```

→

hey, honey!
Hey, honey!

▼ str.casefold() - converts string into lower case

```
b = 'hey, honey!'  
print(b.upper())  
print(b.casefold())
```

→

HEY, HONEY!
hey, honey!

▼ str.center() - returns a centered string

```
b = 'hey, honey!'  
print(b.center(110, '.'))
```

→

.....hey, honey!.....

- ▼ str.count() - returns the number of times a specified value occurs in a string

```
b = 'hey, honey!'
print(b.count('h'))
print(b.count('y'))
```

→

2
2

- ▼ str.encode() - returns an encoded version of a string

```
getdefaultencoding()
# utf-8

my_string = 'кот cat'

type(my_string)
# str

my_string.encode()
# b'\xd0\xba\xd0\xbe\xd1\x82 cat'

my_string.encode('ascii')
# UnicodeDecodeError

my_string.encode('ascii', errors='ignore')
# b' cat'

my_string.encode('ascii', errors='replace')
# b'??? cat'

my_string.encode('ascii', errors='xmlcharrefreplace')
# b'ком cat'

my_string.encode('ascii', errors='backslashreplace')
```

```

# b'\u043a\u043e\u0442 cat'

my_string.encode('ascii', errors='namereplace')

# \N{CYRILLIC SMALL LETTER KA}\N{CYRILLIC SMALL LETTER O}\N{CYRILLIC SMALL LETTER TE} cat'

surrogated = '\udcd0\udcba\udcd0\udcbe\udcd1\udc82 cat'

surrogated.encode()

# UnicodeEncodeError

surrogated.encode(errors='surrogateescape')

# \xd0\xba\xd0\xbe\xd1\x82 cat'

surrogated.encode(errors='surrogatepass')

# \xed\xb3\x90\xed\xb2\xba\xed\xb3\x90\xed\xb2\xbe\xed\xb3\x91\xed\xb2\x82 cat'

```

- ▼ str.endswith() - returns True if the string ends with the specified value

```

b = 'hey, honey!'
print(b.endswith('!'))
print(b.endswith('honey!'))
print(b.endswith('world!'))

```

→

True

True

False

- ▼ str.expandtabs() - sets the tab size of the string

```

header_table = 'Name!\tRace\tWeapon\tMount'
person1 = 'Tiana\tElf\tSword\tHell Horse'
print(header_table.expandtabs(15))
print(person1.expandtabs(15))

```

→

Name!	Race	Weapon	Mount
Tiana	Elf	Sword	Hell Horse

▼ str.find() - searches the string for a specified value and returns its position

```
lyrics = "I'm completely gone from your mind\nSoon as you realize\nYou let me go  
lighter than air"  
word = lyrics.find('air')  
print(lyrics)  
print('''\nThe word "air" is number''' +str(word)+"in the poem.")
```

→

I'm completely gone from your mind
Soon as you realize
You let me go lighter than air

The word "air" is number82in the poem.

▼ str.format() - formats specified values in a string

```
def cashier(x):  
    return """Here's your check for ${}.""".format(x)  
  
print(cashier(5))
```

→

Here's your check for \$5.

▼ str.format_map() - formats specified values in a string

```
point = {'x':4, 'y':-5}  
print('{x} {y}'.format_map(point))  
  
point = {'x':4, 'y':-5, 'z': 0}  
print('{x} {y} {z}'.format_map(point))
```

→

```
4 -5  
4 -5 0
```

The `format_map()` method is similar to `str.format(mapping)` except that `str.format(**mapping)` creates a new dictionary whereas `str.format_map(mapping)` doesn't.**

- ▼ `str.index()` - searches the string for a specified value and returns its position

```
quote = "Everything is hard before it's easy."
print(quote.find('easy'))
```

→

31

- ▼ `str.isalnum()` - returns True if all characters in a string are alphanumeric

```
year = "1917"
event = "1917 revolution"
print(year.isalnum())
print(event.isalnum())
```

→

True

False

- ▼ `str.isalpha()` - returns True if all characters in a string are in the alphabet

```
army = "army"
red_army = "red army"
event = "1917 revolution"
print(army.isalpha())
print(red_army.isalpha())
print(event.isalpha())
```

→

True

False

False

- ▼ `str.isascii()` - returns True if all characters in a string are ASCII characters

```
".isascii() # True  
'wz'.isascii() # True  
'w z'.isascii() # True  
'фы'.isascii() # False  
'wzфы'.isascii() # False
```

- ▼ str.isdecimal() - returns True if all characters in a string are decimal

```
year = "1917"  
event = "1917 revolution"  
print(year.isdecimal())  
print(event.isdecimal())
```

→

```
True  
False
```

- ▼ str.isdigit() - returns True if all characters in a string are digits

```
".isdigit() # False  
' '.isdigit() # False  
'!@#'.isdigit() # False  
'abc'.isdigit() # False  
'123'.isdigit() # True  
'abc123'.isdigit() # False
```

- ▼ str.isidentifier() - returns True if the string is an identifier

```
'continue'.isidentifier() # True  
'cat'.isidentifier() # True  
'function_name'.isidentifier() # True  
'ClassName'.isidentifier() # True  
'_'.isidentifier() # True  
'number1'.isidentifier() # True
```

```
'1st'.isidentifier() # False
```

```
'*'.isidentifier() # False
```

- ▼ str.islower() - returns True if all characters in teh string are lower case

```
year = "1917"
event = "1917 revolution"
rev = "revolution"
print(year.islower())
print(event.islower())
print(rev.islower())
```

→

False

True

True

- ▼ str.isnumeric() - returns True if all characters in the string are numeric

```
".isnumeric() # False
```

```
'a'.isnumeric() # False
```

```
'0'.isnumeric() # True
```

```
'10'.isnumeric() # True
```

```
'½'.isnumeric() # True
```

```
'XII'.isnumeric() # True
```

- ▼ str.isprintable() - returns True if all characters in the string are printable

```
".isprintable() # True
```

```
' '.isprintable() # True
```

```
'1'.isprintable() # True
```

```
'a'.isprintable() # True
```

```
'█'.isprintable() # False (Group Separator)
```

```
'█'.isprintable() # False (Escape)
```

- ▼ str.isspace() - retirns True if all characters in the string are whitespaces

```
year = "19 17"
space = "      "
print(year.isspace())
print(space.isspace())
```

→

False

True

- ▼ str.istitle() - returns True if the string follows the rules of a title

```
title1 = "The Lonely Cat"
title2 = "The lonely cat"
title3 = "THE LONELY CAT"
title4 = "the lonely cat"
print(title1.istitle())
print(title2.istitle())
print(title3.istitle())
print(title4.istitle())
```

→

True

False

False

False

- ▼ str.isupper() - returns True if all the characters in the string are upper case

```
title1 = "The Lonely Cat"
title2 = "The lonely cat"
title3 = "THE LONELY CAT"
title4 = "the lonely cat"
print(title1.isupper())
print(title2.isupper())
print(title3.isupper())
print(title4.isupper())
```

→

```
False  
False  
True  
False
```

▼ `iterable.join()` - converts elements of an iterable into a string

```
list = ["like", "love", "hate"]  
x = '-'.join(list)  
print(x)  
tuple = ("like", "love", "hate")  
y = '-'.join(tuple)  
print(y)  
dict = {"me":"love", "you":"like", "she":"hate"}  
z = '-'.join(dict)  
print(z)
```

→

```
like-love-hate  
like-love-hate  
me-you-she
```

▼ `str.ljust()` - returns a left justified version of the string

```
quote = "All we need is love"  
print(quote.ljust(30, 'e'))
```

→

```
All we need is loveeeeeeeeeeee
```

▼ `str.lower()` - converts the string into lower case

```
quote = "All we need is love"  
print(quote.lower())
```

→

```
all we need is love
```

▼ `str.lstrip()` - returns a left trim version of the string

```
quote = "All we need is love"
print(quote.lstrip('All'))
print((quote.lstrip('we'))))
print((quote.lstrip('need'))))
print((quote.lstrip('is'))))
```

→

```
we need is love
All we need is love
All we need is love
All we need is love
```

- ▼ `srt.maketrans()` - returns a translation table to be used in translations

```
trans_table = str.maketrans({
    'a': 'b',
    'r': 't',
    'z':None,
})
# {97: 'b', 114: 't', 122: None}

trans_table = str.maketrans('ar', 'bt', 'z')
# {97: 98, 114: 116, 122: None}

'arroz'.translate(trans_table)
# 'btto'
```

- ▼ `str.partition()` - returns a tuple where the string is parted into three parts

```
quote = "All we need is love"
print(quote.partition(' '))
print((quote.partition('need'))))
empty = ''
print(empty.partition('.')))
numb = '.2'
print((numb.partition('.'))))
```

→

```
('All', '', 'we need is love')
('All we ', 'need', ' is love')
(' ', ' ', ' ')
(' ', '!', '2')
```

- ▼ str.replace() - returns a string where a specified value is replaced with a specified value

```
quote = "All we need is love"
print(quote.replace('love', 'money'))
```

→

All we need is money

- ▼ str.rfind() - searches the string for a specified value and returns the last position of where it was found

```
quote = "All we need is love"
print(quote.rfind('e'))
```

→

18

- ▼ str.rindex() - searches the string for a specified value and returns the last position of where it was found

```
quote = "All we need is love"
print(quote.rindex('l'))
```

→

15

- ▼ str.rjust() - returns a right justified version of the string

```
quote = "All we need is love"
a = quote.rjust(75)
print(a, '- "The Beatles" sang.')
```

→

All we need is love - "The Beatles" sang.

- ▼ str.rpartition() - returns a tuple where the string is parted into three parts

```
quote = "All we need is love"
print(quote.rpartition('love'))
```

→

('All we need is ', 'love', '')

- ▼ str.rsplit() - splits the string at the specified separator, and returns a list

```
quote = "All we need is love"
print(quote.rsplit('is'))
```

→

['All we need ', ' love']

- ▼ str.rstrip() - returns a right trim version of the string

```
quote = "All we need is love"
print(quote.rstrip('e'))
```

→

All we need is lov

- ▼ str.split() - splits the string at the specified separator, and returns a list

```
quote = "All you and I need is love and health"
print(quote.split('and'))
```

→

['All you ', ' I need is love ', ' health']

▼ str.splitlines() - splits the string at line breaks and returns a list

```
quote = "All you and I need \nis love and health"  
print(quote.splitlines())  
print(quote.splitlines(keepends=True))
```

→

```
['All you and I need ', 'is love and health']  
['All you and I need \n', 'is love and health']
```

▼ str.startswith() - returns True if a string starts with a specified value

```
quote = "All you and I need \nis love and health"  
print(quote.startswith('All'))
```

→

```
True
```

▼ str.strip() - returns a trimmed version of a string

```
print('love'.strip('l'))  
print('love'.strip('e'))  
print('love'.strip('o'))
```

→

```
ove  
lov  
love
```

▼ str.swapcase() - swaps cases

```
print('No love allowed'.swapcase())  
print('cAPS LOCK LEAP'.swapcase())
```

→

nO LOVE ALLOWED

Caps lock leap

- ▼ str.title() - converts the first character of each word into upper case

```
print('No love allowed'.title())
print('cAPS LOCK LEAP'.title())
```

→

No Love Allowed

Caps Lock Leap

- ▼ str.translate() - returns a translated string

```
trans_table = str.maketrans({'л':'к', 'д':'а', '-':None})
replaced = 'лошка -ест сметану'.translate(trans_table)
print(replaced)
```

→

кошка ест сметану

- ▼ str.upper() - converts a string into upper case

```
song = 'Love the way \nyou lie'
print(song.upper())
```

→

LOVE THE WAY

YOU LIE

- ▼ str.zfill() - fills the string with a specified number of 0 values at the beginning

```
number = '13'
specific_number = number.zfill(10)
print(specific_number)
```

→

0000000013

Boolean Type

Also called as Boolean Context and “Truthiness”, Boolean Type means that the value of an object of this type may be True or False.

They are keywords, or expressions, more precisely.

▼ E.g.:

```
def love_test(petals):
    if petals % 2 == 0:
        return True
    else:
        return False

print(love_test(5))
print(type(love_test(5)))
```

→

```
False
<class 'bool'>
```

True = 1, False = 0

▼ Counting “the” in a poem

```
>>> lines = """\
... He took his vorpal sword in hand;
...       Long time the manxome foe he sought-
... So rested he by the Tumtum tree
...       And stood awhile in thought.
...
>>> line_list = lines.splitlines()
>>> "the" in line_list[0]
False
>>> "the" in line_list[1]
True
>>> 0 + False + True # Equivalent to 0 + 0 + 1
1
>>> ["the" in line for line in line_list]
[False, True, True, False]
>>> False + True + True + False
2
>>> len(line_list)
```

```
4
>>> 2/4
0.5
```

Shorter:

```
>>> lines="""\
... He took his vorpal sword in hand;
...     Long time the manxome foe he sought-
... So rested he by the Tumtum tree
...         And stood awhile in thought.
... """ .splitlines()
>>> sum("the" in line.lower() for line in lines) / len(lines)
0.5
```

truth tables

short-circuit evaluation

The operator “**not**” (“not True” or “not False”). Another operator - “**and**” - takes two arguments (A and B), and means that the expression is False until both A and B are True. One more operator - “**or**” - is True until either A or B is True.

Comparison operators: “==” (equality); “!=” (inequality), “<” (less than, strict); “>” (greater than, strict); “<=” (less or equal, not strict), “>=” (greater or equal, not strict).

Can’t compare: str and int, dict, set. List, tuple can be compared as they are ordered lexicographically.

The operators “is” and “is not” check if A is the same object as B, or not.

The operator “in” checks an object from membership in somewhere (e.g. array).

One can chain operators: 1<3<7. Or mixed: a = 0 \ a is a<1 → True.

“None” is always False. All numbers are True, except for 0.

▼ One more value, or object in this case, evaluates to `False`, and that is if you have an object that is made from a class with a `__len__` function that returns `0` or `False`:

```
class myclass():
    def __len__(self):
```

```
return 0  
  
myobj = myclass()  
print(bool(myobj))
```

Functions

▼ Parametr or argument?

From a function's perspective:

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called.

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

▼ Arbitrary arguments

▼ If ikd how many args - *args:

```
def guests(*guests):  
    print(guests[0]+" was the first to come to my BD party!")  
  
guests("Jamie", "Anne", "Lucy", "Christoph")
```

→

Jamie was the first to come to my BD party!

▼ Keyword args

```
def family(mum, dad, me):  
    print("Hi! My name is "+me+". My mum's name is "+mum+". My dad's name is "+dad  
+".")  
family(mum = "Emily", dad="Peter", me="Clair")
```

→

Hi! My name is Clair. My mum's name is Emily. My dad's name is Peter.

▼ Arbitrary keyword args

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: `**` before the parameter name in the function definition.

▼ This way the function will receive a *dictionary* of arguments, and can access the items accordingly:

```
def detective_dairy(**info):
    print("I've got a client whose last name is"+info["lname"]+".\n")
    print("She told me to spy after her husband, "+info["hname"]+", as she thought he was cheating on her.\n")
    print("Yesterday morning I saw him leave for work, hiding "+info["present1"]+
" under suit coat.\n")
    print("In the afternoon he left his office and went to "+info["place1"]+
".\n")
    print("He came back with "+info["present2"]+".\n")
    print("Then he got a taxi to "+info["place2"]+".\n")
    print("He met "+info["person1"]+" at the entrance and greeted her as a close friend.\n")
    print("In the restaurant "+info["person2"]+" joined them.\n")
    print("After that they went to "+info["place3"]+".\n")
    print("Soon after the wife came to the same place. She realized that her husband prepared "+info["event"]+" for "+info["person3"]+".\n")
    print("That was an interesting story, even though I had to work more as "+info["proff"]+".")

print("\nHere's the first story:\n.....\n")
detective_dairy(lname="Clarson", hname="Ray", present1="a red box", place1="a flower shop", present2="a huge bouquet of red roses", place2="restaurant", person1="a red-haired girl", person2="a brightly dressed brunette", place3="his house", event="a birthday party", person3="her", proff="a photographer")

print("\nHere's the second story:\n.....\n")
detective_dairy(lname="Lanson", hname="Sam", present1="a black box", place1="a store", present2="a medium package", place2="a park", person1="a man in a blue suit", person2="a man in a white suit", place3="his house", event="a trap", person3="her", proff="a police officer")

print("\n.....")
print("\nIndeed, how a few details can change the whole picture! Fascinating, isn't it?")
```

→

Here's the first story:.....

I've got a client whose last name isClarson.

She told me to spy after her husband, Ray, as she thaught he was cheating on her.

Yesterday morning I saw him leave for work, hiding a red box under suit coat.

In the afternoon he left his office and went to a flower shop.

He came back with a huge bouquet of red roses.

Then he got a taxi to restaurant.

He met a red-haired girlat the entrance and greeted her as a close friend.

In the restaurant a brightly dressed brunettejoined them.

After that they went to his house.

Soon after the wife came to the same place. She realized that her husband prepared a birthday partyforher.

That was an interesting story, even though I had to work more as a photographer.

Here's the second story:.....

I've got a client whose last name isLanson.

She told me to spy after her husband, Sam, as she thaught he was cheating on her.

Yesterday morning I saw him leave for work, hiding a black box under suit coat.

In the afternoon he left his office and went to a store.

He came back with a medium package.

Then he got a taxi to a park.

He met a man in a blue suitat the entrance and greeted her as a close friend.

In the restaurant a man in a white suitjoined them.

After that they went to his house.

Soon after the wife came to the same place. She realized that her husband prepared a trapforher.

That was an interesting story, even though I had to work more as a police officer.

.....

Indeed, how a few details can change the whole picture! Fascinating, isn't it?

▼ Default parametr value

```
def attendant_country(country=" Russia"):
    print("- Hi! Where are you from?")
    print("- Hi! I'm from"+country+". Great conference, isn't it?")
    if country == "Russia":
        print("- Yes! And which city are you from?")
    else:
        print("- Indeed! How do you like our city? Have you been sightseeing?")

print("-----Dialogue 1-----")
attendant_country()
print()
print("-----Dialogue 2-----")
attendant_country(country=" the USA")
```

→

-----Dialogue 1-----

- Hi! Where are you from?
- Hi! I'm from Russia. Great conference, isn't it?
- Yes! And which city are you from?

-----Dialogue 2-----

- Hi! Where are you from?
- Hi! I'm from the USA. Great conference, isn't it?
- Indeed! How do you like our city? Have you been sightseeing?

▼ Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

▼ E.g. if you send a List as an argument, it will still be a List when it reaches the function

```
def words(wordlist):
    for x in wordlist:
        print(x)

vocab = ["a cat", "a mouse", "a house"]
words(vocab)
```

→

a cat
a mouse
a house

▼ return

▼ To let a function return a value, use the `return` statement:

```
def myfunc(x):
    return x**5

print(myfunc(1))
print(myfunc(2.1))
print(myfunc(-3))
```

→

1
40.84101000000001
-243

▼ pass

`function` definitions cannot be empty, but if you for some reason have a `function` definition with no content, put in the `pass` statement to avoid getting an error.

▼ Recursion

Recursion means that a function can call itself, which enables us to loop through data to reach a result.

▼ Example

In this example, `tri_recursion()` is a function that we have defined to call itself ("recurse"). We use the `k` variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

```
def tri_recursion(k):
    if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
    else:
        result = 0
```

```
    return result

print("Recursion Example Results")
tri_recursion(6)
```

→

Recursion Example Results

```
1
3
6
10
15
21
```



▼ Explanation

Try tracing the function with a pencil and paper. In this case, the print statement inside the function may be a bit misleading.

Consider this part of the program,

```
if(k>0):
    result = k+tri_recursion(k-1)
    ...
```

From here,

```
tri_recursion(6) = 6 + tri_recursion(5)
```

So to get the result for `tri_recursion(6)` we must get the result of `tri_recursion(5)`. Following this logic, the problem reduces to:

```
tri_recursion(6)
= 6 + tri_recursion(5)
= 6 + 5 + tri_recursion(4)
= 6 + 5 + 4 + tri_recursion(3)
= 6 + 5 + 4 + 3 + tri_recursion(2)
= 6 + 5 + 4 + 3 + 2 + tri_recursion(1)
= 6 + 5 + 4 + 3 + 2 + 1 + tri_recursion(0)
```

Now notice that 0 is not greater than 0 so the program moves to the body of the else clause:

```
else:
    result = 0
...
```

Which means `tri_recursion(0) = 0`. Therefore:

```
tri_recursion(6)
= 6 + 5 + 4 + 3 + 2 + 1 + tri_recursion(0)
= 6 + 5 + 4 + 3 + 2 + 1 + 0
= 21
```

Points to note:

1. In running this program, `k` is never equal to `1`, infact it is impossible.
2. It is misleading to think of control flow in terms of "the compiler moving across a program". The compiler doesn't do anything during execution (JIT is a different matter). It is better to think in terms of control flow / order of execution in procedural languages, equationally in functional programming and relations in logic programming.

▼ Lambda

It's an anonymous function, that can take any number of arguments, but can only have one expression.

▼ Syntax: `lambda arguments : expression`

```
x = lambda a, b : a/b
print(x(2, 3))
print(x(5, -10))
```

→

0.6666666666666666

-0.5

▼ Lambda can be used inside another function

```
def myfunc(n):
    return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

→

22

33

In Python there are some **Built-in functions:**

▼ Math

▼ `abs()` - returns absolute value of a number

```
print(abs(-26))
print(abs(0.2))
print(abs(83))
print(abs(-3.09))
```

→

26

0.2

83

3.09

- ▼ `divmod()` - returns quotient and remainder of integer division

```
print(divmod(45, 5))
print(divmod(33, 2))
```

→

(9, 0)

(16, 1)

- ▼ `max()` - returns the largest of given arguments or items in an iterable

```
list = [-3, 0, 1, 2, 4, 5, 7, 8]
print(max(list))
```

→

8

- ▼ `min()` - returns the smallest of given arguments or items in an iterable

```
list = ['lol', 'love']
list1 = ['a', 'b']
list2 = ['1', '2', 'a', 'b']
print(min(list))
print(min(list1))
print(min(list2))
```

→

lol

a

1

```
list3 = [1, 2, 'a', 'b']
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print(min(list3))

TypeError: '<' not supported between instances of 'str' and 'int'

```
tuple = (1, 2, 'love', 'a')
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3, in <module>
print(min(tuple))

TypeError: '<' not supported between instances of 'str' and 'int'

```
dict = {1:"May", "cat":13, 2:19, "mouse":"grey"}
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print(min(dict))

TypeError: '<' not supported between instances of 'str' and 'int'

```
tuple = ('love', 'a')
dict = {"mouse":"grey"}
print(min(tuple))
print(min(dict))
```

→

a

mouse

▼ pow() - raises a number to a power

```
print(pow(4, 2))
print(pow(-0.3, 3))
```

→

```
16  
-0.02699999999999996
```

▼ round() - rounds floating-point value

```
print(round(5.3))  
print(round(5.5))  
print(round(5.6))  
print(round(-5.3))  
print(round(-5.5))  
print(round(-5.6))
```

→

```
5  
6  
6  
-5  
-6  
-6
```

▼ sum() - sums the items of an iterable

```
list = [1, 2, -3, 5]  
tuple = (9, 8, 0.5)  
print(sum(list))  
print(sum(tuple))
```

→

```
5  
17.5
```

```
dict = {1:2, 7:4}  
print(sum(dict))
```

→ sums only keys

8

▼ Type Conversion

- ▼ `ascii()` - returns a string containing a printable representation of an object
- ▼ `bin()` - converts an integer to a binary string

```
print(bin(0))
print(bin(1))
print(bin(2))
print(bin(3))
print(bin(4))
print(bin(5))
print(bin(6))
print(bin(7))
print(bin(8))
print(bin(9))
print(bin(10))
print(bin(11))
print(bin(-1))
```

→

```
0b0
0b1
0b10
0b11
0b100
0b101
0b110
0b111
0b1000
0b1001
0b1010
0b1011
-0b1
```

- ▼ `bool()` - converts an argument to a Boolean value

```
print(bool(1))
print(bool(0))
print(bool(5))
print(bool(-0.4))
print(bool('lol'))
```

```
list = [0]
print(bool(list))
list1 = []
print(bool(list1))
```

→

True
False
True
True
True
True
False

- ▼ chr() - returns representation of character given by integer argument

```
print(chr(65))
print(chr(93))
```

→

A
J

Given an integer from 0 to 255, chr() returns str with the relevant character from ASCII table.

- ▼ complex() - returns a complex number constructed from arguments

complex() returns a number with two parts - real and imaginary (with j)

```
print(complex(2, -0.3))
print(complex(4))
```

→

(2-0.3j)
(4+0j)

- ▼ float() - returns a floating-point object constructed from a number or string

```
print(float(9))
print(float(0))
print(float("12.3"))
```

→

9.0

0.0

12.3

▼ `hex()` - converts an integer to a hexadecimal string

```
print(hex(5))
print(hex(277))
print(hex(-48))
print(hex(0))
```

→

0x5

0x115

-0x30

0x0

▼ `int()` - returns an integer object constructed from a number or a string

```
print(int('23'))
print(int(2.3))
print(int(-2.3))
```

→

23

2

-2

▼ `oct()` - converts an integer to an octal string

```
print(oct(8))
print(oct(-8))
```

→

0o10
-0o10

▼ **ord()** - returns integer representation of a character

The opposite to **chr()**

```
print(ord('A'))  
print(ord('a'))  
print(ord('z'))  
print(ord('Z'))
```

→

65
97
122
90

▼ **repr()** - returns a string containing a printable representation of an object

```
list = (0.1, -2)  
print(repr(1))  
print(repr('Lol'))  
print(repr(list))
```

→

1
'Lol'
(0.1, -2)

▼ **str()** - returns a string version of an object

```
list = (0.1, -2)  
dict = {"panda":23, 1:"bear"}  
print(str(1))  
print(repr(dict))  
print(str(list))  
print(type(str(dict)))
```

```
→  
1  
{'panda': 23, 1: 'bear'}  
(0.1, -2)  
<class 'str'>
```

- ▼ `type()` - returns the type of an object or creates a new type object

```
dict = {"panda":23, 1:"bear"}  
print(type(dict))
```

```
→  
<class 'dict'>
```

`type()` can also be used to create a class faster. These two methods create the same object:

```
>>>class Foo(object):  
...      bar =True
```

and

```
Foo = type('Foo', (), {'bar':True})
```

- ▼ Iterables and Iterators

- ▼ `all()` - returns True if all elements of an iterable are true

```
list = [0, 1, 2, "lol"]  
list1 = [1, 2, "lol"]  
print(all(list))  
print(all(list1))
```

```
→  
False  
True
```

- ▼ `any()` - returns True if any elements of an iterable are true

```
list = [0]
list1 = [0, 1, 2, "lol"]
print(any(list))
print(any(list1))
```

→

False

True

- ▼ `enumerate()` - returns a list of tuples containing indices and values from an iterable

```
list = [-1, 0, 1, 2, 3, 4]
for i in enumerate(list):
    print(i)
```

→

(0, -1)
(1, 0)
(2, 1)
(3, 2)
(4, 3)
(5, 4)

The first element of a tuple is index of an element of an iterable, and the second - its value.

- ▼ `filter()` - filters elements from an iterable

E.g.:

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
print(list(filter(lambda x: x[0].lower() in 'aieou', creature_names)))
```

→

['Odrey']

Read as:

I need to print a list

that is a result of function filter()

that is performed with usage of function lambda(),

where every element of an iterable is represented as x;

x[0] enables lambda() to go through every first letter of every word in the list creature_names,

lower - makes the first letters written in lowers case so that they were the same as our example ‘aieou’;

in the end I mention th elist where I need there operations to be done.

▼ iter() - returns an iterator object

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
iterated_creature_names = iter(creature_names)
print(next(iterated_creature_names))
print(next(iterated_creature_names))
print(next(iterated_creature_names))
print(next(iterated_creature_names))
```

→

Lola

Lilu

Jim

Odrey

The number of iterator object call outs must not exceed the number of objects in the iterable.

▼ len() - returns the length of an object

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
name = "Kiki"
print(len(creature_names))
print(len(name))
```

→

4

4

Doesn't work with int and float.

- ▼ **map()** - applies a function to every item of an iterable

```
ist1 = [1, 2, 3, 4]
mapped_list = list(map(lambda x: x**5-1, list1))
print(mapped_list)
```

→

[4, 9, 14, 19]

```
aquarium_creatures = [
    {"name": "sammy", "species": "shark", "tank number": 11, "type": "fish"},
    {"name": "ashley", "species": "crab", "tank number": 25, "type": "shellfish"},
    {"name": "jo", "species": "guppy", "tank number": 18, "type": "fish"},
    {"name": "jackie", "species": "lobster", "tank number": 21, "type": "shellfish"},
    {"name": "charlie", "species": "clownfish", "tank number": 12, "type": "fish"},
    {"name": "olly", "species": "green turtle", "tank number": 34, "type": "turtle"}
]
def assign_to_tank(aquarium_creatures, new_tank_number):
    def apply(x):
        x["tank number"] = new_tank_number
        return x
    return map(apply, aquarium_creatures)
print(list(assign_to_tank(aquarium_creatures, 42)))
```

→

```
[{"name": "sammy", "species": "shark", "tank number": 42, "type": "fish"}, {"name": "ashley", "species": "crab", "tank number": 42, "type": "shellfish"}, {"name": "jo", "species": "guppy", "tank number": 42, "type": "fish"}, {"name": "jackie", "species": "lobster", "tank number": 42, "type": "shellfish"}, {"name": "charlie", "species": "clownfish", "tank number": 42, "type": "fish"}, {"name": "olly", "species": "green turtle", "tank number": 42, "type": "turtle"}]
```

```
base = [1, 2, 3]
powers = [1, 2, 3]
powered_numbers = list(map(pow, base, powers))
print(powered_numbers)
```

→

[1, 4, 27]

- ▼ next() - retrieves the next item from an iterator

```
numb = iter([1, 3, 5])
print(next(numb))
print(next(numb))
print(next(numb))
```

→

1
3
5

- ▼ range() - generates a range of integer values

```
for item in range(10):
    print(item)
```

→

0
1
2
3
4
5
6
7
8
9

▼ reversed() - returns a reverse iterator

```
numb = [1, 2, 3]
print(list(reversed(numb)))
```

→

[3, 2, 1]

▼ slice() - returns a slice object

```
quote = "Per aspera ad astra"
a = slice(4, 10)
b = slice(0, 3)
print(quote[a])
print(quote[b])
```

→

aspera

Per

▼ sorted() - returns a sorted list from an iterable

```
signs = ["Aries", "Taurus", "Gemini", "Cancer", "Leo", "Virgo", "Libra", \
         "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pieces"]
print(sorted(signs))
```

→

['Aquarius', 'Aries', 'Cancer', 'Capricorn', 'Gemini', 'Leo', 'Libra', 'Pieces', 'Sagittarius',
'Scorpio', 'Taurus', 'Virgo']

▼ zip() - creates an iterator that aggregates elements from iterables

```
signs = ["Aries", "Taurus", "Gemini", "Cancer", "Leo", "Virgo", "Libra", \
         "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pieces"]
dates = ["(21st March - 20th April)", "(21st April - 21st May)", "(22nd May - 21st June)", "(22nd June - 22nd July)", "(23rd July - 23rd August)", "(24th August - 22nd September)", "(23rd September - 23rd October)", "(24th October - 22nd November)", "(23rd November - 21st December)", "(22nd December - 20th January)", "(21st January - 18th February)", "(19th February - 20th March)"]
```

```
for s, d in zip(signs, dates):
    print(s, d)
```

→

Aries (21st March – 20th April)
Taurus (21st April - 21st May)
Gemini (22nd May - 21st June)
Cancer (22nd June - 22nd July)
Leo (23rd July - 23rd August)
Virgo (24th August - 22nd September)
Libra (23rd September - 23rd October)
Scorpio (24th October - 22nd November)
Sagittarius (23rd November - 21st December)
Capricorn (22nd December - 20th January)
Aquarius (21st January - 18th February)
Pisces (19th February - 20th March)

▼ If/elif/else

▼ if...else...

```
French_vocab = {
    "food" : {
        "le fromage":"cheese",
        "le lait":"milk"
    },
    "animals" : {
        "le chat":"a cat",
        "l'elephant":"an elephant"
    },
    "time" : {
        "le soir": "tonight",
        "le matin":"this morning"
    }
}

def Antoshka():
    x = input('Введите слово: ')
    if x in French_vocab["animals"] or x in French_vocab["food"] or x in French_vocab["time"]:
        print("О, я знаю это слово! Но я забыл...")
    else:
        print("Это мы не проходили, это нам не задавали...")
```

```
Antoshka()
```

→

[input] le chat

[output] О, я знаю это слово! Но я забыл...

▼ elif

Stands in between:

```
French_vocab = {  
    "food" : {  
        "le fromage": "cheese",  
        "le lait": "milk"  
    },  
    "animals" : {  
        "le chat": "a cat",  
        "l'elephant": "an elephant"  
    },  
    "time" : {  
        "le soir": "tonight",  
        "le matin": "this morning"  
    }  
}  
  
def Antoshka():  
    x = input('Введите слово: ')  
    if x in French_vocab["food"] :  
        print("О, это что-то съедобное!")  
    elif x in French_vocab["animals"] or x in French_vocab["time"]:  
        print("Не знаю, что это, но точно не еда!")  
    else:  
        print("Это мы не проходили, это нам не задавали...")  
  
Antoshka()
```

→

[input] le chat

[output] Не знаю, что это, но точно не еда!

▼ Shorthand if

```
bag_items = ["sun_screen", "bottle_of_water", "magazine", "phone"]
x = input("Есть ли у Насти:")
if x in bag_items: print("Да, у Насти это есть!")
```

▼ Shorthand if...else...

This technique is known as **Ternary Operators**, or **Conditional Expressions**

```
bag_items = ["sun_screen", "bottle_of_water", "magazine", "phone"]
x = input("Есть ли у Насти:")
print("Да, у Насти это есть!") if x in bag_items else print("Такого нет!")
```

▼ Multiple conditions

```
x = int(input("Введите целое число: "))
print("Число больше нуля.") if x>0 else print("Число равно нулю.") if x==0 else print("Число меньше нуля.")
```

▼ Nested if

```
print('Добро пожаловать в нашу игру "Великолепный век!"')
x = input("Выберите пол персонажа: ").title()
if x == "Мужчина":
    y = int(input("Сколько вы задонатили в игру? Введите целое число: "))
    if y >= 1000:
        print("Поздравляем! Вы - султан! Вся власть в ваших руках!")
    else:
        print("Вы - слуга. Стоит задонатить еще, чтобы стать султаном!")
else:
    z = input("Напишите ваше главное качество: ").title()
    while z != "Красивая":
        s = input("Выберите другое качество: ").title()
        z //="Красивая"
    print("Подравляем! Вы в гареме!")
```

▼ pass statement

`if` statements cannot be empty, but if you for some reason have an `if` statement with no content, put in the `pass` statement to avoid getting an error.

```
a = 33  
b = 200  
if b > a:  
    pass
```

▼ While loop

▼ With the while loop we can execute a set of statements as long as a condition is true.

Note: remember to increment i, or else the loop will continue forever.

```
a = 0  
while a!=10:  
    print(a)  
    a = a+1
```

→

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

▼ break - stops the loop at the condition

```
a = 0  
while a!=10:  
    print(a)  
    if a == 5:  
        break  
    a = a+1
```

→

```
0  
1  
2  
3  
4  
5
```

- ▼ continue - stops prev. iteration and continues with the next one

```
i = 1  
while i < 6:  
    i *= 2  
    if i == 3:  
        continue  
    print(i)
```

→

```
2  
4  
8
```

- ▼ else

```
i = 1  
while i < 6:  
    print(i)  
    i *= 2  
else:  
    print("The end")
```

→

```
1  
2  
4  
The end
```

x//= “something”- another way to break the loop

▼ for loop

▼ loop

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    print(i)
```

→

a floor
windows
a door
a ceiling
walls

▼ break

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    print(i)
    if i == "a door":
        break
```

→

a floor
windows
a door

▼ continue

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    if i == "windows":
        continue
    print(i)
```

→

a floor
a door

a ceiling
walls

▼ range()

```
for x in range(10):
    print(x*3)
    x = x+1
```

→

0
3
6
9
12
15
18
21
24
27

▼ else

```
for x in range(5, 10, 2):
    print(x*3)
    x = x+1
else:
    print("Finish")
```

→

15
21
27
Finish

```
for x in range(5, 10):
    if x == 8: break
    print(x)
```

```
    else:  
        print("Finish")
```

→

5
6
7

▼ nested loops

▼ The "inner loop" will be executed one time for each iteration of the "outer loop":

```
girls = ["Sara", "Rachel", "Emily", "Lola"]  
boys = ["Sam", "Michael", "Jack", "Brandon"]  
  
def couples():  
    for x in girls:  
        for y in boys:  
            print(str(x), "+", str(y), "= a nice couple!")  
  
couples()
```

→

Sara + Sam = a nice couple!
Sara + Michael = a nice couple!
Sara + Jack = a nice couple!
Sara + Brandon = a nice couple!
Rachel + Sam = a nice couple!
Rachel + Michael = a nice couple!
Rachel + Jack = a nice couple!
Rachel + Brandon = a nice couple!
Emily + Sam = a nice couple!
Emily + Michael = a nice couple!
Emily + Jack = a nice couple!
Emily + Brandon = a nice couple!
Lola + Sam = a nice couple!
Lola + Michael = a nice couple!

Lola + Jack = a nice couple!
Lola + Brandon = a nice couple!

▼ pass

`for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.

```
for x in [0, 1, 2]:  
    pass
```

Composite Data Type

▼ `bytearray()` - creates and returns an object of the `bytearray` class (also known as “binary type”)

```
>>>b = bytearray(b'hello world!')  
>>>b  
bytearray(b'hello world!')  
>>>b[0]  
104  
>>>b[0] = b'h'  
Traceback (most recent call last):  
  File "", line 1, in  
    b[0] = b'h'  
TypeError: an integer is required  
>>>b[0] = 105  
>>>b  
bytearray(b'iello world!')  
>>>for i in range(len(b)):  
...     b[i] += i  
...>>>b  
bytearray(b'ifnos%}vzun,')
```

▼ `bytes()` - creates and returns a `bytes` object (similar to `bytearray` but immutable) (also known as “binary type”)

```
>>>b 'bytes'  
b'bytes'  
>>>'Байты'.encode('utf-8')  
b'\xd0\x91\xd0\xbd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'  
>>>bytes('bytes', encoding = 'utf-8')  
b'bytes'
```

```
>>>bytes([50, 100, 76, 72, 41])  
b'2dLH')
```

- ▼ dict() - creates a dict object (also known as “mapping type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",  
Taurus = "21st April - 21st May",  
Gemini = "22nd May - 21st June",  
Cancer = "22nd June - 22nd July",  
Leo = "23rd July - 23rd August",  
Virgo = "24th August - 22nd September",  
Libra = "23rd September - 23rd October",  
Scorpio = "24th October - 22nd November",  
Sagittarius = "23rd November - 21st December",  
Capricorn = "22nd December - 20th January",  
Aquarius = "21st January - 18th February",  
Pieces = "19th February - 20th March")  
print(zodiac_signs)  
print(zodiac_signs['Leo'])
```

→

```
{'Aries': '21st March – 20th April', 'Taurus': '21st April - 21st May', 'Gemini': '22nd May - 21st June', 'Cancer': '22nd June - 22nd July', 'Leo': '23rd July - 23rd August', 'Virgo': '24th August - 22nd September', 'Libra': '23rd September - 23rd October', 'Scorpio': '24th October - 22nd November', 'Sagittarius': '23rd November - 21st December', 'Capricorn': '22nd December - 20th January', 'Aquarius': '21st January - 18th February', 'Pieces': '19th February - 20th March'}  
23rd July - 23rd August
```

- ▼ frozenset() - creates a frozenset object (it's like set, but immutable)

```
b = frozenset('qwerty')  
print(b.add(1))
```

→

```
Traceback (most recent call last):  
File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 2, in <module>  
print(b.add(1))  
AttributeError: 'frozenset' object has no attribute 'add'
```

▼ `list()` - creates a list object (also known as “sequence type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(list(zodiac_signs))
```

→

```
['Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo', 'Libra', 'Scorpio', 'Sagittarius', 'Capricorn',
'Aquarius', 'Pieces']
```

▼ `object()` - creates a new featureless object

You cannot add new properties or methods to this object. This object is the base for all classes, it holds the built-in properties and methods which are default for all classes.

```
test = object()

print(type(test))
print(dir(test))
```

→

```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__',
 '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__']
```

In the program, we have used `type()` to get the type of the object. Similarly, we have used `dir()` to get all the attributes. These attributes (properties and methods) are common to instances of all Python classes.

▼ `set()` - creates a set object (also known as “set type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(set(zodiac_signs))
```

→

```
{'Scorpio', 'Gemini', 'Pieces', 'Virgo', 'Aquarius', 'Leo', 'Sagittarius', 'Aries', 'Libra', 'Taurus',
'Cancer', 'Capricorn'}
```

▼ `tuple()` - creates a tuple object (also known as “sequence type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(tuple(zodiac_signs))
```

→

```
('Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo', 'Libra', 'Scorpio', 'Sagittarius', 'Capricorn',
'Aquarius', 'Pieces')
```

▼ `*memoryview` (also known as “binary type”) -

▼ `** range` (also known as “sequence type”) -

Arrays

▼ List (lis') - an ordered mutable collection of objects of various types (str and int cannot be in one list in Python3).

▼ Let's create a list:

- Way 1:

```
a = list('Morning')
print(a)
```

→

['M', 'o', 'r', 'n', 'i', 'n', 'g']

- Way 2:

```
a = [1, 2, 3]
print(a)
```

→

[1, 2, 3]

- Way 3:

```
list_generator = [c*3 for c in range(3)]
print(list_generator)
```

→

[0, 3, 6]

▼ List Methods

▼ list.append(x) - adds an elements to the end of a list

```
sample = [1, 2, 3]
sample.append(4)
```

```
print(sample)
sample.append(5)
print(sample)
```

→

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
```

- ▼ `list.extend(L)` - extends a list, adding elements from the list L to the end

```
sample = [1, 2, 3]
another_sample = [4, 5, 6]
sample.extend(another_sample)
print(sample)
print(another_sample)
```

→

```
[1, 2, 3, 4, 5, 6]
[4, 5, 6]
```

- ▼ `list.insert(i, x)` - pastes an x-element to the i-position in the list

```
sample = [1, 2, 3]
print(sample)
sample.insert(3, 4)
print(sample)
```

→

```
[1, 2, 3]
[1, 2, 3, 4]
```

- ▼ `list.remove(x)` - deletes the first x-element (if none - ValueError)

```
sample = [1, 2, 3, 2]
print(sample)
sample.remove(2)
print(sample)
```

→

[1, 2, 3, 2]

[1, 3, 2]

- ▼ list.pop(i) - deletes the element with i-index and returns the list. If list.pop() - deletes the last element

```
sample = [1, 2, 3, 2]
print(sample)
sample.pop(2)
print(sample)
```

→

[1, 2, 3, 2]

[1, 2, 2]

- ▼ list.index(x, [start [, end]]) - returns the index of the first x-element (searches from start to end)

```
sample = [1, 2, 2, 2, 3, 2]
print(sample)
print(sample.index(2))
```

→

[1, 2, 2, 2, 3, 2]

1

- ▼ list.count(x) - returns quantity of x-elements

```
sample = [1, 2, 2, 2, 3, 2]
print(sample)
print(sample.count(2))
```

→

[1, 2, 2, 2, 3, 2]

4

- ▼ list.sort(key=function/ None, reverse = True/ False) - sorts out the list according to the function given

```
sample_list = ['Rose', 'Lily', 'Sharon', 'Mio']
sample_list.sort()
print(sample_list)
sample_list.sort(reverse=True)
print(sample_list)

def sample_func (x):
    return len(x)
sample_list.sort(key=sample_func, reverse=True)
print(sample_list)
```

→

```
['Lily', 'Mio', 'Rose', 'Sharon']
['Sharon', 'Rose', 'Mio', 'Lily']
['Sharon', 'Rose', 'Lily', 'Mio']
```

▼ `list.reverse()` - returns a reversed list

```
sample_list = ['а р о з а у п а л а н а р у к у а з о р а']
sample_list.reverse()
print(sample_list)
sample_list.reverse()
print(sample_list)
```

→

```
['а р о з а у п а л а н а р у к у а з о р а']
['а р о з а у п а л а н а р у к у а з о р а']
```

▼ `list.copy()` - makes a shallow copy of a list

```
# mixed list
prime_numbers = [2, 3, 5]

# copying a list
numbers = prime_numbers.copy()

print('Copied List:', numbers)

# Output: Copied List: [2, 3, 5]
```

Another way to make a copy is to use the built-in method `list()`

```
thislist = ["apple", "banana", "cherry"]
```

```
mylist = list(thislist)
```

```
print(mylist)
```

▼ list.clear() - clears up a list

```
sample_list = ['а р о з а у п а л а н а р у к у а з о р а']  
sample_list.reverse()  
print(sample_list)  
sample_list.clear()  
print(sample_list)
```

→

```
['а р о з а у п а л а н а р у к у а з о р а']  
[]
```

List methods change the list itself, so we don't need to use a variable for the changed list.

▼ Looping a list

▼ Loop through a list

```
thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
    print(x)
```

→

```
apple  
banana  
cherry
```

▼ Loop by index

```
thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
    print(thislist[i])
```

→

```
apple  
banana  
cherry
```

▼ While Loop

```
thislist = [1, 2, 3, 4, 5, 6 ,7 ,8, 9]  
i = 0  
while i < len(thislist):  
    print(thislist[i])  
    i = i + 1
```

```
→  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

▼ Looping Using List Comprehension

```
thislist = ["apple", "banana", "cherry"]  
[print(x) for x in thislist]
```

```
→  
apple  
banana  
cherry
```

▼ List Comprehencion

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Syntax: newlist = [*expression* for *item* in *iterable* if *condition* == True]

The *condition* is like a filter that only accepts the items that evaluate to `True`.

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list.

▼ Without:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

→

`['apple', 'banana', 'mango']`

▼ With:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

→

`['apple', 'banana', 'mango']`

▼ More examples:

```
#Accept only numbers lower than 5:
newlist = [x for x in range(10) if x < 5]
print(newlist)
```

→

`[0, 1, 2, 3, 4]`

```

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x.upper() for x in fruits] #Set the values in the new list to upper case:

print(newlist)

newlist = ['hello' for x in fruits] #Set all values in the new list to 'hello':

print(newlist)

newlist = [x if x != "banana" else "orange" for x in fruits] #Return "orange" instead of "banana":

print(newlist)

```

→

```

['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
['hello', 'hello', 'hello', 'hello', 'hello']
['apple', 'orange', 'cherry', 'kiwi', 'mango']

```

▼ Nested

▼ Nested loops can also be used in a list comprehension expression. To obtain list of all combinations of items from two lists:

```

print([{x:y} for x in range(1, 15, 2) for y in range(1, 25, 2)])
def cons(word):
    result = [char for char in word if char not in ["a", "e", "i", "o", "u"]]
    print(result)
cons(word="siblings")
cons(word="country")

```

→

```

[{:1: 1}, {:1: 3}, {:1: 5}, {:1: 7}, {:1: 9}, {:1: 11}, {:1: 13}, {:1: 15}, {:1: 17}, {:1: 19}, {:1: 21}, {:1: 23}, {:3: 1}, {:3: 3}, {:3: 5}, {:3: 7}, {:3: 9}, {:3: 11}, {:3: 13}, {:3: 15}, {:3: 17}, {:3: 19}, {:3: 21}, {:3: 23}, {:5: 1}, {:5: 3}, {:5: 5}, {:5: 7}, {:5: 9}, {:5: 11}, {:5: 13}, {:5: 15}, {:5: 17}, {:5: 19}, {:5: 21}, {:5: 23}, {:7: 1}, {:7: 3}, {:7: 5}, {:7: 7}, {:7: 9}, {:7: 11}, {:7: 13}, {:7: 15}, {:7: 17}, {:7: 19}, {:7: 21}, {:7: 23}, {:9: 1}, {:9: 3}, {:9: 5}, {:9: 7}, {:9: 9}, {:9: 11}, {:9: 13}, {:9: 15}, {:9: 17},

```

```
{9: 19}, {9: 21}, {9: 23}, {11: 1}, {11: 3}, {11: 5}, {11: 7}, {11: 9}, {11: 11},  
{11: 13}, {11: 15}, {11: 17}, {11: 19}, {11: 21}, {11: 23}, {13: 1}, {13: 3},  
{13: 5}, {13: 7}, {13: 9}, {13: 11}, {13: 13}, {13: 15}, {13: 17}, {13: 19},  
{13: 21}, {13: 23}]  
['s', 'b', 'T', 'n', 'g', 's']  
['c', 'n', 't', 'r', 'y']
```

▼ Join Lists

▼ Concatenation

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
animals = ["koala", "bear", "flying fox", "rabbit", "turtle"]  
  
animals_and_their_food = fruits + animals  
print(animals_and_their_food)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'koala', 'bear', 'flying fox', 'rabbit', 'turtle']
```

▼ .append() + for

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
more_fruits = ["pear", "dragon fruit", "pineapple"]  
  
for x in more_fruits:  
    fruits.append(x)  
  
print(fruits)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'pear', 'dragon fruit', 'pineapple']
```

▼ .extend()

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
more_fruits = ["pear", "dragon fruit", "pineapple"]
```

```
fruits.extend(more_fruits)  
print(fruits)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'pear', 'dragon fruit', 'pineapple']
```

▼ Tuple (tuple) - an ordered immutable collection of objects (like list but immutable).

Immutability makes a tuple more secure than a list: it can't be changed neither intentionally (which is also sad) nor unintentionally (which is great).

Moreover, a tuple weights less, it is processed faster , and it can be used as a key in a dictionary.

You are allowed to **add tuples to tuples (or multiply a tuple)**, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple.

▼ Let's create a tuple

```
a = tuple('Hello, world!')  
b = ('hello', 'world')  
c = 'hello', 'world'  
print(a)  
print(b)  
print(c)  
print(type(a))  
print(type(b))  
print(type(c))
```

→

```
('H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')  
('hello', 'world')  
('hello', 'world')  
<class 'tuple'>  
<class 'tuple'>  
<class 'tuple'>
```

▼ Operations with tuples

```
b = ('hello', 'world')  
print(b[0])
```

```
b[0] = 'new'
```

→

hello

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3, in <module>

```
b[0] = 'new'
```

TypeError: 'tuple' object does not support item assignment

(we can take an element from a tuple, but not change or delete it; still we can delete a whole tuple)

```
b = ['hello', 'world']
print(tuple(b))
print(type(tuple(b)))
```

→

('hello', 'world')

<class 'tuple'>

```
b = ('hello', 'world')
print(list(b))
print(type(list(b)))
print(b)
print(type(b))
```

→

['hello', 'world']

<class 'list'>

('hello', 'world')

<class 'tuple'>

```
b = ('hello', 'world')
a = list(b)
a[0] = 'new'
print(tuple(a))
```

```
print(type(tuple(a)))
print(type(a))
```

→

```
('new', 'world')
<class 'tuple'>
<class 'list'>
```

▼ Tuple methods

- ▼ tuple.count() - returns a number of times a specified value occurs in a tuple

```
b = ('hello', 'world', 'hello')
print(b.count('hello'))
print(b.index('world'))
```

→

```
2
1
```

- ▼ tuple.index() - searches the tuple for a specified value and returns its position

```
b = ('hello', 'world', 'hello')
print(b.count('hello'))
print(b.index('world'))
```

→

```
2
1
```

▼ Unpacking a Tuple

In Python, we are also allowed to extract the values back into variables. This is called "unpacking":

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
```

```
print(green)
print(yellow)
print(red)
```

→

apple
banana
cherry

Note: *the number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.*

▼ Loop Tuples

▼ Loop through a tuple

```
butterflies = ("monarch", "pharaoh", "papillon")
for x in butterflies:
    print(x)
```

→

monarch
pharaoh
papillon

▼ Loop by index

```
butterflies = ("monarch", "pharaoh", "papillon")
for x in range(len(butterflies)):
    print(butterflies[x])
```

→

monarch
pharaoh
papillon

▼ While Loop

```
butterflies = ("monarch", "pharaoh", "papillon")
i = 0
while i < len(butterflies):
    print(butterflies[i])
    i = i+1
```

→

monarch
pharaoh
papillon

▼ Set (set) - an unordered collection of unique objects.

Set items are unchangeable, but you can remove items and add new items. Also, set items are unindexed, here's why there's no duplicate values in a set.

▼ Let's make a set

```
a = set()
print(a)
b = set("Morning!")
print(b)
c = {1, 2, 3, 4}
print(c)
d = {i for i in range(10)}
print(d)
e = {}
print(type(e))
```

→

{'o', 'n', 'M', 'i', 'g', '!', 'r'}
{1, 2, 3, 4}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
<class 'dict'>

▼ Access items

You cannot access items in a set by referring to an index or a key. But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.

▼ for loop

```
butterflies = {"monarch", "pharaoh", "papillon"}  
for x in butterflies:  
    print(x)
```

→

pharaoh
monarch
papillon

▼ in keyword

```
butterflies = {"monarch", "pharaoh", "papillon"}  
print("pharaoh" in butterflies)
```

→

True

▼ Loop set

```
colors = {'red', 'green', 'blue', 'pink'}  
for x in colors:  
    print(x)
```

→

blue
green
red
pink

Sets are handy when we need to get rid of repetitions:

```
a = ['cat', 'cat', 'mouse']  
print(a)  
print(set(a))
```

→

```
['cat', 'cat', 'mouse']
{'mouse', 'cat'}
```

frozenset - like set, but immutable

▼ Set Methods:

- ▼ a.union(b) - returns a set that is a merge of a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'fish', 'turtle', 'hamster'}
c = a.union(b)
print(c)
```

→

```
{'fish', 'hamster', 'cat', 'mouse', 'dog', 'turtle'}
```

- ▼ a.update(b) - adds elements from b to a (not only set, any iterable)

```
a = {'cat', 'dog', 'mouse'}
b = {'fish', 'turtle', 'hamster'}
a.update(b)
print(a)
```

→

```
{'dog', 'fish', 'mouse', 'cat', 'hamster', 'turtle'}
```

- ▼ a.intersection(b) - returns a set that is an intersection of a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.intersection(b)
print(c)
```

→

```
{'cat'}
```

- ▼ `a.intersection_update(b)` - leaves in the set a only those elements that are present in the set b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.intersection_update(b)
print(a)
```

→

{'cat'}

- ▼ `a.difference(b)` - returns the difference between a and b (those elements that are present in a but not in b)

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.difference(b)
print(c)
```

→

{'dog', 'mouse'}

- ▼ `a.difference_update(b)` - deletes from a elements from b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.difference_update(b)
print(a)
```

→

{'mouse', 'dog'}

- ▼ `a.symmetric_difference(b)` - returns symmetric difference between a and b (elements, present in a or b, but not in both)

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.symmetric_difference(b)
print(c)
```

→

```
{'hamster', 'dog', 'turtle', 'mouse'}
```

- ▼ `a.symmetric_difference_update(b)` - puts in a symmetric difference between a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.symmetric_difference_update(b)
print(a)
```

→

```
{'hamster', 'dog', 'mouse', 'turtle'}
```

- ▼ `a.issubset(b)` - returns True if a is a subset of b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
print(a.issubset(b))
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(c.issubset(d))
```

→

```
False
```

```
True
```

- ▼ `a.issuperset(b)` - returns True if b is a subset of a

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(d.issuperset(c))
```

→

```
True
```

- ▼ `a<b` - equal to $a \leq b$ and $a \neq b$

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(c<d)
```

→

True

- ▼ a>b - equal to a≥b and a≠b

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(d>c)
```

→

True

- ▼ set.add() - adds an element to the set

```
c = {'cat'}
c.add('turtle')
print(c)
```

→

{'turtle', 'cat'}

- ▼ set.clear() - removes all elements from the set

```
s = {'mouse', 'cat', 'house'}
print(s)
s.clear()
print(s)
```

→

{'mouse', 'house', 'cat'}
set()

- ▼ set.copy() - returns a copy of the set

```
s = {'mouse', 'cat', 'house'}
print(s.copy())
d = s.copy()
print(d)
```

→

```
{'mouse', 'house', 'cat'}
{'mouse', 'house', 'cat'}
```

▼ **set.discard()** - removes the specified element

```
s = {'mouse', 'cat', 'house'}
s.discard('mouse')
print(s)
```

→

```
{'house', 'cat'}
```

▼ **set.isdisjoint()** - returns whether two sets have an intersection (True if nothing in common)

```
s = {'mouse', 'cat', 'house'}
s1 = {'mouse', 'cat', 'castle'}
print(s1.isdisjoint(s))
s2 = {'house'}
s3 = {'mouse', 'cat', 'castle'}
print(s2.isdisjoint(s3))
```

→

False

True

▼ **set.pop()** - removes a random element from the set

```
s = {'mouse', 'cat', 'house'}
s.pop()
print(s)
s.pop()
print(s)
```

```
s.pop()  
print(s)
```

→

```
{'cat', 'mouse'}  
{'mouse'}  
set()
```

▼ **set.remove()** - removes a specified element from the set

```
s = {'mouse', 'cat', 'house'}  
s.remove('mouse')  
print(s)
```

→

```
{'cat', 'house'}
```

▼ **Dictionary** (dict) - unoredered collections of various values that have keys. One can get access to a value by using a key, and vice versa. Dict are also called “associative massives” or “hash-tables”. No duplicates (no two items with the same key).

▼ Let's create a dict

```
d = {} #using a literal  
print(d)  
d1 = {1:"Moon", 2:"Earth"} #using a literal  
print(d1)  
d2 = dict(race='elf', weapon='magic') #using dict function  
print(d2)  
d3 = dict([(1,'dollar'),(2,'ruble')]) #using a literal  
print(d3)  
d4 = dict.fromkeys(['b', 'a'], 100) #using fromkeys method  
print(d4)  
d5 = {x: x**3 for x in range(7)} #using a dictionary generator  
print(d5)
```

→

```
{}  
{1: 'Moon', 2: 'Earth'}
```

```
{'race': 'elf', 'weapon': 'magic'}  
{1: 'dollar', 2: 'ruble'}  
  
{'b': 100, 'a': 100}  
  
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216}
```

▼ Add items

```
thisdict = {  
    "brand": "Gucci",  
    "item": "bag",  
    "storage": 3  
}  
thisdict["color"] = "red"  
print(thisdict)
```

→

```
{"brand": 'Gucci', 'item': 'bag', 'storage': 3, 'color': 'red'}
```

▼ Remove items

Apart from `.pop()` and `.popitem()`, you can use `del` to delete a specific item from a dict:

```
thisdict = {  
    "brand": "Gucci",  
    "item": "bag",  
    "storage": 3  
}  
del thisdict["storage"]  
print(thisdict)
```

→

```
{"brand": 'Gucci', 'item': 'bag'}
```

`del` can also delete the whole dict

▼ Let's get a value and a key from a dict, and change values

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1[1])  
d1[2] = 'Mars'  
print(d1)
```

→

Moon

{1: 'Moon', 2: 'Mars'}

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1['Moon'])
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 2, in <module>

print(d1['Moon'])

KeyError: 'Moon'

▼ Loop a dict

▼ get keys

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",  
    "le chat": "a cat",  
    "le soir": "the evening"  
}  
  
for x in French_vocab:  
    print(x)
```

→

le fromage

le lait

le chat

le soir

▼ get values

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",
```

```
"le chat":"a cat",
"le soir":"the evening"
}

for x in French_vocab:
    print(French_vocab[x])
```

→

cheese
milk
a cat
the evening

▼ .values()

```
French_vocab = {
    "le fromage":"cheese",
    "le lait":"milk",
    "le chat":"a cat",
    "le soir":"the evening"
}

for x in French_vocab.values():
    print(x)
```

→

cheese
milk
a cat
the evening

▼ .keys()

```
French_vocab = {
    "le fromage":"cheese",
    "le lait":"milk",
    "le chat":"a cat",
    "le soir":"the evening"
}

for x in French_vocab.keys():
    print(x)
```

→

le fromage
le lait
le chat
le soir

▼ .items()

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",  
    "le chat": "a cat",  
    "le soir": "the evening"  
}  
  
for x, y in French_vocab.items():  
    print(x, y)
```

→

le fromage cheese
le lait milk
le chat a cat
le soir the evening

▼ Copy a dict

Use .copy() or built-in func. dict()

▼ Nested dict

A dictionary can contain dictionaries, this is called nested dictionaries.

```
French_vocab = {  
    "food" : {  
        "le fromage": "cheese",  
        "le lait": "milk"  
    },  
    "animals" : {  
        "le chat": "a cat",  
        "l'elephant": "an elephant"  
    },  
    "time" : {  
        "le soir": "tonight",  
        "le matin": "this morning"
```

```

        }
    }

print(French_vocab)
print(French_vocab["time"])
for x in French_vocab:
    print(x)
for x in French_vocab["animals"].values():
    print(x)

```

→

```

{'food': {'le fromage': 'cheese', 'le lait': 'milk'}, 'animals': {'le chat': 'a cat', "l'elephant": "an elephant"}, 'time': {'le soir': 'tonight', 'le matin': 'this morning'}}}
{'le soir': 'tonight', 'le matin': 'this morning'}
food
animals
time
a cat
an elephant

```

▼ Dict Methods

▼ dict.clear() - clears up the dict

```

d1 = {1:"Moon", 2:"Earth"}
print(d1)
d1.clear()
print(d1)

```

→

```

{1: 'Moon', 2: 'Earth'}
{}

```

▼ dict.copy() - returns a copy of the dict

```

d1 = {1:"Moon", 2:"Earth"}
d2 = d1.copy()
print(d2)
print(d1)

```

→

```
{1: 'Moon', 2: 'Earth'}  
{1: 'Moon', 2: 'Earth'}
```

- ▼ dict.get(key, [default]) - returns the value of the key, if none - returns default (None)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.get(1))  
print(d1.get("Mars"))  
print(d1.get(3))
```

→

```
Moon  
None  
None
```

- ▼ dict.items() - returns key - value pairs that are in the dict

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.items())
```

→

```
dict_items([(1, 'Moon'), (2, 'Earth')])
```

- ▼ dict.keys() - returns keys

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.keys())
```

→

```
dict_keys([1, 2])
```

- ▼ dict.pop(key [, default]) - deletes the key and returns the value, if none - returns default (exception)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1)
```

```
print(d1.pop(1))
print(d1)
```

→

{1: 'Moon', 2: 'Earth'}

Moon

{2: 'Earth'}

- ▼ dict.popitem() - deletes and returns key-value pair, if {} - KeyError (no order)

```
d1 = {1:"Moon", 2:"Earth"}
print(d1)
print(d1.popitem())
print(d1)
print(d1.popitem())
print(d1)
print(d1.popitem())
print(d1)
```

→

{1: 'Moon', 2: 'Earth'}

(2, 'Earth')

{1: 'Moon'}

(1, 'Moon')

{}

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 7, in <module>

print(d1.popitem())

KeyError: 'popitem(): dictionary is empty'

- ▼ dict.setdefault(key[, default]) - returns the value of the key, if none - creates a key with default value (None)

```
d1 = {1:"Moon", 2:"Earth"}
print(d1.setdefault(1))
print(d1.setdefault(3))
```

→

Moon

None

- ▼ `dict.update([other])` - updates the dict by creating key-value pairs from [other]. Existing keys change. Returns None (not a new dict)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1)  
d1.update({3:"Venus"})  
print(d1)  
d1.update({3:"Mars"})  
print(d1)
```

→

```
{1: 'Moon', 2: 'Earth'}  
{1: 'Moon', 2: 'Earth', 3: 'Venus'}  
{1: 'Moon', 2: 'Earth', 3: 'Mars'}
```

- ▼ `dict.values()` - returns values

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.values())
```

→

```
dict_values(['Moon', 'Earth'])
```

Classes and Objects

A **class** is like an object constructor, “a blueprint” for creating new objects (or subclasses). A class has features (attributes) and actions (methods) that objects of this class (or subclasses) inherit. A class can be created during runtime, and changed anytime.

An **object** - basically anything in Python (and everything has its class, hidden in the core of the language).

An object has:

- a state - represented by attributes, reflects the properties of an object;

- some behaviour - represented by methods of an object, reflects the response of an object to other objects;
- an identity - object is given a unique name which enables one object to interact with other objects.

When an object is created (declared), we say that a class has been instantiated.

▼ Let's create a class

```
class MyClass:  
    x = 5  
  
print(MyClass)
```

→

<class 'main.MyClass'>

▼ Let's create an object

```
class MyClass:  
    x = 5  
  
print(MyClass)  
  
p1 = MyClass()  
print(p1.x)
```

→

<class 'main.MyClass'>

5

▼ The `__init__()` Function

Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created

▼ Example

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
person1 = Person("Julie", 16)  
  
print(person1.name)  
print(person1.age)
```

→

Julie

16

The `__init__()` function is called automatically every time the class is being used to create a new object.

▼ Object Methods

Methods in objects are functions that belong to the object.

▼ Example

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def my_name_is(self):  
        print("Hello, my name is " + self.name + ".")  
  
    def im_age(self):  
        print("I'm " + str(self.age) + " years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()
```

▼ The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong to the class. It does not have to be named `self`, you can call it whatever you like, but it has to be the first parameter of any function in the class.

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(per):  
        print("Hello, my name is " + per.name + ".")  
  
    def im_age(per):  
        print("I'm " + str(per.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()
```

→

Hello, my name is Julie.

I'm 16 years old.

▼ Modify Object Properties

You can modify properties on objects like this:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(person1):  
        print("Hello, my name is " + person1.name + ".")  
  
    def im_age(person1):  
        print("I'm " + str(person1.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()  
  
person1.age = 17  
print((person1.age))
```

→

Hello, my name is Julie.

I'm 16 years old.

17

▼ Delete Object Properties

You can delete properties on objects by using the `del` keyword:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(person1):  
        print("Hello, my name is " + person1.name + ".")  
  
    def im_age(person1):  
        print("I'm " + str(person1.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()  
  
del person1.age  
print((person1.age))
```

→

Hello, my name is Julie.

I'm 16 years old.

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 17, in <module>

print((person1.age))

AttributeError: 'Person' object has no attribute 'age'

▼ Delete Objects

You can delete objects by using the `del` keyword:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):
```

```

some_self.name = name
some_self.age = age

def my_name_is(person1):
    print("Hello, my name is " + person1.name + ".") 

def im_age(person1):
    print("I'm " + str(person1.age) +" years old.")

person1 = Person("Julie", 16)
person1.my_name_is()
person1.im_age()

del person1

print(person1)

```

→

Hello, my name is Julie.

I'm 16 years old.

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 18, in <module>

print((person1))

NameError: name 'person1' is not defined

▼ The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the `pass` statement to avoid getting an error.

▼ Example

```

class Person:
    pass
print(Person)

```

→

<class 'main.Person'>

```

class Alien:
    pass
print(Alien)

```

→

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3

print(Alien)

^

IndentationError: expected an indented block

Class Methods:

- ▼ classmethod() - returns a class method for a function
- ▼ delattr() - deletes an attribute from an object
- ▼ getattr() - returns the value of a named attribute of an object
- ▼ hasattr() - returns True if an object has a given attribute
- ▼ isinstance() - determines whether an object is an instance of a given class
- ▼ issubclass() - determines whether an object is an instance of a given subclass
- ▼ property() - returns a property value of a class
- ▼ setattr() - sets the value of a named attribute of an object
- ▼ super() - returns a proxy object that delegates method calls to parent or sibling class

Input/ Output

- ▼ format() - converts a value to a formatted representation
- ▼ input() - reads input from the console
- ▼ open() - opens a file and returns a file object
- ▼ print() - prints to a text stream or the console

Variables, References, and Scope

- ▼ dir() - returns a list of names in current local scope or a list of object attributes
- ▼ globals() - returns a dictionary representing the current global symbol table
- ▼ id() - returns the identity of an object
- ▼ locals() - updates and returns a dictionary representing current local symbol table

- ▼ `vars()` - returns `_dict_` attribute for a module, class, or object

Miscellaneous

- ▼ `callable()` - returns `True` if object appears callable
- ▼ `compile()` - compiles source into a code or AST object
- ▼ `eval()` - evaluates a Python expression
- ▼ `exec()` - implements dynamic execution of Python code
- ▼ `hash()` - returns the hash value of an object
- ▼ `help()` - invokes the built-in help system
- ▼ `memoryview()` - returns a memory view object
- ▼ `staticmethod()` - returns a static method for a function
- ▼ `__import__()` - invoked by the `import` statement

Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class inherits from base class, and child class - from parent class.

- ▼ Example

```
class Person:  
    def __init__(self, fname, lname, gender):  
        self.firstname = fname  
        self.lastname = lname  
        self.gender = gender  
  
    def greeting(self):  
        print("Hello! My name is "+self.firstname+" "+self.lastname+".")  
  
class Student(Person):  
    def __init__(self, fname, lname, gender, faculty):  
        Person.__init__(self, fname, lname, gender)  
        self.university = "Harvard"  
        self.faculty = faculty  
  
    def myuniversity(self):  
        print("I study in "+self.university+".")
```

```

def myfaculty(self):
    print("I study at the "+self.faculty+" faculty.")

def accomodation(self):
    if self.gender == "male":
        print("I live in the male dormitory.")
    else:
        print("I live in the female dormitory.")

print("Here's a man who looks like being in his forties:\n")
y = Person("Finn", "Olleyson", "male")
y.greeting()
print()
print("Here's a girl who looks like being in her twenties:\n")
x = Student("Emma", "Tompson", "female", "Computer Science")
x.greeting()
x.myuniversity()
x.myfaculty()
x.accomodation()

```

→

Here's a man who looks like being in his forties:

Hello! My name is Finn Olleyson.

Here's a girl who looks like being in her twenties:

Hello! My name is Emma Tompson.

I study in Harvard.

I study at the Computer Science faculty.

I live in the female dormitory.

The child's `__init__()` function **overrides** the inheritance of the parent's `__init__()` function.

To keep the inheritance of the parent's `__init__()` function, add a call to the parent's `__init__()` function.

`super()` function makes the child class inherit all the methods and properties from its parent without naming the parent class.

Iterators

An iterator is an object that contains a countable number of values. It can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods `__iter__()` and `__next__()`.

Iterator vs Iterable

Lists, tuples, dictionaries, and sets (and str) are all iterable objects. They are iterable *containers* which you can get an iterator from.

- ▼ All these objects have a `iter()` method which is used to get an iterator:

```
tup = ("tiger", "lion", "panther")
myit = iter(tup)

print(next(myit))
print(next(myit))
print(next(myit))
```

→

tiger
lion
panther

- ▼ Loop

```
tup = ("tiger", "lion", "panther")
for x in tup:
    print(x)
```

→

tiger
lion
panther

- ▼ Class Iterator

To create an object/class as an iterator you have to implement the methods `__iter__()` and `__next__()` to your object.

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
```

```

        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))

```

→

1
2
3
4
5

▼ StopIteration

To prevent the iteration to go on forever, we can use the `StopIteration` statement.

In the `__next__()` method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

```

class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        if self.a <= 20:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

```

```
for x in myiter:  
    print(x)
```

→

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Sope

“Уровень вложенности” in Russian. It can be local or global.

Modules

Consider a module to be the same as a code library. A file containing definition of functions, classes, variables, constants or any other Python object, even some executable code you want to include in your application.

▼ Create a module

To create a module just save the code you want in a file with the file extension `.py`

▼ Use a module

Use the `import` statement (import a module). When using a function from a module, use the syntax: *module_name.function_name*

▼ Variables in a module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc).

▼ Naming and renaming

You can use any name.py. You can create an alias when you import a module, by using the `as` keyword (import some_module as sm).

▼ Built-in modules

To see the whole list print: `help('modules')`. Below there are some frequently used modules.

▼ os module

Has functions to perform many tasks of operating system.

▼ `mkdir()` - creates a new directory.

A new directory corresponding to path in string argument in the function will be created. If we open D drive in Windows explorer we should notice tempdir folder created.

```
>>> import os  
>>> os.mkdir("d:\\tempdir")
```

▼ `chdir()`- changes current working directory

```
>>> import os  
>>> os.chdir("d:\\temp")
```

▼ `getcwd()` - returns name of a current working directory

```
>>> os.getcwd()  
'd:\\temp'
```

Directory paths can also be relative. If current directory is set to D drive and then to temp without mentioning preceding path, then also current working directory will be changed to d:\temp.

```
>>> os.chdir("d:\\\\")
>>> os.getcwd()
'd:\\\\'
>>> os.chdir("temp")
>>> os.getcwd()
'd:\\\\temp'
```

In order to set current directory to parent directory use ".." as the argument to chdir() function.

```
>>> os.chdir("d:\\\\temp")
>>> os.getcwd()
'd:\\\\temp'
>>> os.chdir("..")
>>> os.getcwd()
'd:\\\\'
```

- ▼ rmdir() - removes a specific directory either with absolute or relative path (it shouldn't be current working directory, an it shouldn't be empty)

```
>>> os.chdir("tempdir")
>>> os.getcwd()
'd:\\\\tempdir'
>>> os.rmdir("d:\\\\temp")
PermissionError: [WinError 32] The process cannot access the file because it
is being used by another process: 'd:\\\\temp'
>>> os.chdir("..")
>>> os.rmdir("temp")
```

- ▼ listdir() - returns a list of all files in a directory

```
>>> os.listdir("c:\\\\Users")
['acer', 'All Users', 'Default', 'Default User', 'desktop.ini', 'Public']
```

- ▼ random module

Contains randomization functions.

Python uses a pseudo-random generator based upon Mersenne Twister algorithm that produces 53-bit precision floats. Functions in this module depend on pseudo-random number generator function `random()` which generates a random float number between 0.0 and 1.0.

- ▼ `random.random` - returns a random float number between 0.0 and 1.0 (no args).

```
>>> import random  
>>> random.random()  
0.755173688207591
```

- ▼ `random.randint()` - returns a random number between the given numbers

```
>>> random.randint(1,100)  
91
```

- ▼ `random.randrange()` - returns a random element from the range (start, stop, step args)

```
>>> random.randrange(0,101,10)  
40
```

- ▼ `random.choice()` - returns a randomly selected eleemnt from a sequence object (e.g. str, list, tuple).If empty - IndexError.

```
>>> import random  
>>> random.choice('computer')  
'o'
```

- ▼ `random.shuffle()` - randomly reorders elements in a list

```
>>> numbers=[12,23,45,67,65,43]  
>>> random.shuffle(numbers)  
>>> numbers  
[23, 12, 43, 65, 67, 45]
```

▼ math module

This module presents commonly required mathematical functions (trigonometric, representation, logarithmic, angle conversion functions).

In addition, two mathematical constants are also defined in this module (Pie, Euler's number, etc.).

▼ Trigonometric functions

▼ radians() - converts angle in degrees to radians

```
>>> math.radians(30)
0.5235987755982988
```

▼ degrees()- converts angle in radians in degrees

```
>>> math.degrees(math.pi/6)
29.99999999999996
```

▼ math.log() - returns natural logarithm of given number; natural logarithm is calculated on the base e

math.log10(): returns base-10 logarithm or standard logarithm of given number.

```
>>> math.log10(10)
1.0
```

▼ math.exp() - returns a float number after raising e to given number : exp(x) is equivalent to e**x

```
>>> math.e**10
22026.465794806703
```

▼ math.pow() - receives two args, raises first to the second, returns the result

```
>>> math.pow(4,4)  
256.0
```

▼ `math.sqrt()` - computes square root of given number

```
>>> math.sqrt(100)  
10.0
```

▼ Representation functions

▼ `math.ceil()` - approximates the given number to the smallest integer greater than or equal to the given float number

```
>>> math.ceil(4.5867)  
5
```

▼ `math.floor()` - returns the largest integer less than or equal to the given number

```
>>> math.floor(4.5687)  
4
```

▼ time module

▼ `time()` - returns current system time in ticks. The ticks is number of seconds elapsed after epoch time i.e. 12.00 am, January 1, 1970.

```
>>> time.time()  
1544348359.1183174
```

▼ `localtime()` - translates time in ticks in a time tuple notation.

```
>>> tk=time.time()  
>>> time.localtime(tk)  
time.struct_time(tm_year=2018, tm_mon=12, tm_mday=9, tm_hour=15, tm_min=11, tm_sec=25, tm_wday=6, tm_yday=343, tm_isdst=0)
```

- ▼ `asctime()` - returns a readable format of local time

```
>>> tk=time.time()  
>>> tp=time.localtime(tk)  
>>> time.asctime(tp)  
'Sun Dec  9 15:11:25 2018'
```

- ▼ `ctime()` - returns string representation of system's current time

```
>>> time.ctime()  
'Sun Dec  9 15:17:40 2018'
```

- ▼ `sleep()` - halts current program execution for a specified duration in seconds

```
>>> time.ctime()  
'Sun Dec  9 15:19:14 2018'  
>>> time.sleep(20)  
>>> time.ctime()  
'Sun Dec  9 15:19:34 2018'
```

▼ sys module

Provides functions and variables used to manipulate different parts of the Python runtime environment.

- ▼ `sys.argv` - returns the list of command line args passed to a Python script Item[0] - script name. Rest of the arguments are stored at subsequent indices.

Here is a Python script (`test.py`) consuming two arguments from command line.

```
import sys  
print ("My name is {}. I am {} years old".format(sys.argv[1], sys.argv[2]))
```

This script is executed from command line as follows:

```
C:\python37>python tmp.py Anil 23  
My name is Anil. I am 23 years old
```

▼ sys.exit - causes program to end and return to either Python console or command prompt. It is used to safely exit from program in case of exception.

▼ sys.maxsize - returns the largest integer a variable can take

```
>>> import sys  
>>> sys.maxsize  
9223372036854775807
```

▼ sys.path - an environment variable that returns search path for all Python modules

```
>>> sys.path  
['', 'C:\\\\python37\\\\Lib\\\\idlelib', 'C:\\\\python37\\\\python36.zip', 'C:\\\\python3  
7\\\\DLLs', 'C:\\\\python37\\\\lib', 'C:\\\\python37', 'C:\\\\Users\\\\acer\\\\AppData\\\\Ro  
aming\\\\Python\\\\Python37\\\\site-packages', 'C:\\\\python37\\\\lib\\\\site-packages']
```

▼ sys.stdin , sys.stdout , sys.stderr - file objects used by the interpreter for standard input, output and errors. stdin is used for all interactive input (Python shell). stdout is used for the output of print() and of input(). The interpreter's prompts and error messages go to stderr.

▼ sys.version - displays a string containing version number of current Python interpreter

▼ collections module

Provides alternatives to built-in container data types such as list, tuple and dict.

▼ namedtuple() - a factory function that returns object of a tuple subclass with named fields. Any valid Python identifier may be used for a field name except for names starting with an underscore.

```
collections.namedtuple(typename, field-list)
```

The typename parameter is the subclass of tuple. Its object has attributes mentioned in field list. These field attributes can be accessed by lookup as well as by its index.

Following statement declares a employee namedtuple having name, age and salary as fields

```
>>> import collections  
>>> employee=collections.namedtuple('employee', [name, age, salary])  
To create a new object of this namedtuple  
>>> e1=employee("Ravi", 251, 20000)
```

Values of the field can be accessible by attribute lookup

```
>>> e1.name  
'Ravi'
```

Or by index

```
>>> e1[0]  
'Ravi'
```

▼ `OrderedDict()` - Ordered dictionary is similar to a normal dictionary. However, normal dictionary the order of insertion of keys in it whereas ordered dictionary object remembers the same. The key-value pairs in normal dictionary object appear in arbitrary order.

```
>>> d1={}
>>> d1['A']=20
>>> d1['B']=30
>>> d1['C']=40
>>> d1['D']=50
```

We then traverse the dictionary by a for loop,

```
>>> for k,v in d1.items():
print (k,v)

A 20
B 30
D 50
C 40
```

But in case of `OrderedDict` object:

```
>>> import collections  
>>> d2=collections.OrderedDict()  
>>> d2['A']=20  
>>> d2['B']=30  
>>> d2['C']=40  
>>> d2['D']=50
```

Key-value pairs will appear in the order of their insertion.

```
>>> for k,v in d2.items():  
print (k,v)  
A 20  
B 30  
C 40  
D 50
```

▼ deque() - A deque object supports append and pop operation from both ends of a list. It is more memory efficient than a normal list object because in a normal list, removing one of item causes all items to its right to be shifted towards left. Hence it is very slow.

```
>>> q=collections.deque([10,20,30,40])  
>>> q.appendleft(110)  
>>> q  
deque([110, 10, 20, 30, 40])  
>>> q.append(41)  
>>> q  
deque([0, 10, 20, 30, 40, 41])  
>>> q.pop()  
40  
>>> q  
deque([0, 10, 20, 30, 40])  
>>> q.popleft()  
110  
>>> q  
deque([10, 20, 30, 40])
```

▼ statistics module

▼ mean() - calculate arithmetic mean of numbers in a list

```
>>> import statistics  
>>> statistics.mean([2, 5, 6, 9])  
5.5
```

- ▼ median() - returns middle value of numeric data in a list. For odd items in list, it returns value at $(n+1)/2$ position. For even values, average of values at $n/2$ and $(n/2)+1$ positions is returned.

```
>>> import statistics  
>>> statistics.median([1, 2, 3, 8, 9])  
3  
>>> statistics.median([1, 2, 3, 7, 8, 9])  
5.0
```

- ▼ mode() - returns most repeated data point in the list

```
>>> import statistics  
>>> statistics.mode([2, 5, 3, 2, 8, 3, 9, 4, 2, 5, 6])  
2
```

- ▼ stdev() - calculates standard deviation on given sample in the form of list

```
>>> import statistics  
>>> statistics.stdev([1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5])  
1.3693063937629153
```

Some more modules ([TBA](#)):

- ▼ re module ([TBA](#))

Модуль для работы с регулярными выражениями

- ▼ match() - ищет последовательность в начале строки
- ▼ search() - ищет первое совпадение с шаблоном
- ▼ findall() - ищет все совпадения с шаблоном, возвращает результирующие строки в виде списка
- ▼ finditer() - ищет все совпадения с шаблоном, возвращает итератор

- ▼ `compile()` - компилирует регуляторное выражение (к этому объекту затем можно применять все перечисляемые функции)
 - ▼ `fullmatch()` - вся строка должна соответствовать описанному регулярному выражению
 - ▼ `re.sub()` - для замены в строках
 - ▼ `re.split()` - для разделения строки на части
- ▼ **threading module (TBA)**

Thread-based parallelism. This module constructs higher-level threading interfaces on top of the lower level `_tread` module.

- ▼ `active_count()` - return the number of Thread objects currently alive (the returned count is equal to the length of the list returned by `enumerate()`).
- ▼ `current_thread()` - return the current `Thread` object, corresponding to the caller's thread of control. If the caller's thread of control was not created through the `threading` module, a dummy thread object with limited functionality is returned.
- ▼ `excepthook()` - handle uncaught exception raised by `Thread.run()`.

The `args` argument has the following attributes:

- `exc_type`: Exception type.
 - `exc_value`: Exception value, can be `None`.
 - `exc_traceback`: Exception traceback, can be `None`.
 - `thread`: Thread which raised the exception, can be `None`.
- ▼ `__excepthook__` - holds the original value of `threading.excepthook()`. It is saved so that the original value can be restored in case they happen to get replaced with broken or alternative objects.
- ▼ `get_ident()` - return the ‘thread identifier’ of the current thread.
- ▼ `get_native_id()` - return the native integral Thread ID of the current thread assigned by the kernel
- ▼ `enumerate()` - return a list of all `Thread` objects currently active
- ▼ `main_thread()` - Return the main `Thread` object.

- ▼ `settrace(func)` - set a trace function for all threads started from the `threading` module
- ▼ `gettrace()` - get the trace function as set by `settrace()`
- ▼ `setprofile(func)` - Set a profile function for all threads started from the `threading` module
- ▼ `getprofile()` - get the profiler function as set by `setprofile()`
- ▼ `stack_size([size])` - return the thread stack size used when creating new threads
- ▼ `TIMEOUT_MAX()` - The maximum value allowed for the *timeout* parameter of blocking functions (`Lock.acquire()`, `RLock.acquire()`, `Condition.wait()`, etc.)
- ▼ tkinter module
- ▼ csv module
- ▼ pickle module
- ▼ socket module
- ▼ sqlite3 module
- ▼ json module
- ▼ itertools module (все они создают итераторы)
 - ▼ `count(start=, step=)` - returns an iterator of evenly spaced values (infinite iterator)

```
import itertools
x = itertools.count(start=10, step=5)
print(next(x))
print(next(x))
print(next(x))
print(next(x))
```

→

10, 15, 20, 25

```
import itertools
x = itertools.count(start=10, step=5)
for i in x:
    if i>35:
```

```
        break
    else:
        print(i, end=" ")
```

→

10 15 20 25 30 35

```
import itertools
l1 = [1, 2, 3, 4, 5]
l2 = zip(itertools.count(0, 1), l1)
print(list(l2))
```

→

[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]

```
import itertools
l1 = map(lambda x: x**2, itertools.count(0, 1))
for i in l1:
    if i > 60:
        break
    else:
        print(i, end=" ")
```

→

0 1 4 9 16 25 36 49

- ▼ `cycle(obj)` - returns each element from given iterable and saves its copy, when iterating object ends, returns copy - this repetition forms an infinite loop (infinite iterator)

```
import itertools
l1 = [1, 2, 3]
l2 = itertools.cycle(l1)
count = 0
for i in l2:
    if count > 15:
        break
    else:
        print(i, end=" ")
    count += 1
```

→

1 2 3 1 2 3 1 2 3 1 2 3 1

```
import itertools
s = "Go! "
i = itertools.cycle(s)
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
```

→

G
o
!

G
o
!

- ▼ `repeat(obj, times)` - returns the object argument repeatedly (infinite iterator)

```
import itertools
s = "Go! "
i = itertools.repeat(s)
print(next(i))
print(next(i))
print(next(i))
```

→

Go!
Go!
Go!

```
import itertools
iterobj = itertools.repeat("Go!", times=10)
```

```
for i in iterobj:  
    print(i, end=" ")
```

→

Go! Go! Go! Go! Go! Go! Go! Go! Go!

```
import itertools  
powobj = map(pow, range(15), itertools.repeat(2))  
for i in powobj:  
    print(i, end=" ")
```

→

0 1 4 9 16 25 36 49 64 81 100 121 144 169 196

```
import itertools  
l = [1, 2, 3]  
obj = zip(itertools.repeat(5), l)  
print(list(obj))  
obj2 = zip(l, itertools.repeat(7))  
print(list(obj2))
```

→

[(5, 1), (5, 2), (5, 3)]

[(1, 7), (2, 7), (3, 7)]

```
import itertools  
l = ['Lolly', 'Molly', 'Polly']  
obj = zip(itertools.repeat("Hello"), l)  
print(next(obj))  
print(next(obj))  
print(next(obj))
```

→

('Hello', 'Lolly')

('Hello', 'Molly')

('Hello', 'Polly')

- ▼ `accumulate(iterable,func,*initial=None)` - applies a function to successive items in a list (finite iterator)

`functools.reduce()` возвращает только конечное накопленное значение для аналогичной функции.

```
import itertools
#using add and mul operator,so importing operator module
import operator
#using reduce(),so importing reduce() from functools module
from functools import reduce

l1=itertools.accumulate([1,2,3,4,5])
print(list(l1))
```

→

[1, 3, 6, 10, 15]

```
r1=reduce(operator.add,[1,2,3,4,5])
print (r1)
```

→

15

```
l2=itertools.accumulate([1,2,3,4,5],operator.add,initial=10)
print (list(l2))
```

→

[10, 11, 13, 16, 20, 25]

```
l3=itertools.accumulate([1,2,3,5,5],operator.mul)
print (list(l3))
```

→

[1, 2, 6, 30, 150]

```
r2=reduce(operator.mul,[1,2,3,4,5])  
print(r2)
```

→

120

```
l4=itertools.accumulate([2,4,6,3,1],max)  
print (list(l4))
```

→

[2, 4, 6, 6, 6]

```
r3=reduce(max,[2,4,6,3,1])  
print (r3)
```

→

6

```
l5=itertools.accumulate([2,4,6,3,1],min)  
print(list(l5))
```

→

[2, 2, 2, 2, 1]

```
r4=reduce(min,[2,4,6,3,1])  
print (r4)
```

→

1

- ▼ `chain()` - (finite iterator) создает итератор, который возвращает элемент из итерируемого объекта до тех пор, пока он не закончится, а потом переходит к

следующему. Он будет рассматривать последовательности, идущие друг за другом, как одну.(finite iterator)

```
import itertools

l1 = itertools.chain(["cat", "dog"], [5, 7, 9], "LOL")
print(list(l1))
```

→

['cat', 'dog', 5, 7, 9, 'L', 'O', 'L']

▼ `chain.from_iterable(iterable)` - берет один итерируемый объект в качестве входного аргумента и возвращает «склеенный» итерируемый объект, содержащий все элементы входного (finite iterator)

```
import itertools

l1 = itertools.chain.from_iterable(["kot", "kot", "kot"])
print((list(l1)))
```

→

['k', 'o', 't', 'k', 'o', 't', 'k', 'o', 't']

```
l1 = itertools.chain.from_iterable([1, 2, 3])
print((list(l1)))
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print((list(l1)))

TypeError: 'int' object is not iterable

▼ `compress(data, selectors)` - фильтрует элементы *data*, возвращая только те, которые содержат соответствующий элемент в селекторах (*selectors*), стоящих в True. Прекращает выполнение, когда либо данные, либо селекторы закончились. (finite iterator)

```

import itertools

selectors1 = [True, False, True, False]
l1 = itertools.compress([1, 2, 3, 4], selectors1)
print(list(l1))
selectors2 = [False, False]
l2 = itertools.compress([1, 0], selectors2)
print(list(l2))
selectors3 = [True, True]
l3 = itertools.compress([0, 0, 0], selectors3)
print(list(l3))

```

→

[1, 3]

[]

[0, 0]

▼ `dropwhile(predicate, iterable)` - выбрасывает элементы из итерируемого объекта до тех пор, пока предикат (*predicate*) имеет значение `True`, а затем возвращает каждый элемент. Итератор не вернет выходных данных, пока предикат не получит значение `False`.(finite iterator)

```

import itertools

greater_three = itertools.dropwhile(lambda item: item<3, [1, 2, 3, 4])
print(list(greater_three))

```

→

[3, 4]

```

import itertools

return_if_first_odd = itertools.takewhile(lambda item: item%2!=0, [1, 3, 5,
4, 7])
print(list(return_if_first_odd))

return_if_first_even = itertools.takewhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are True to the predicate until it meets False, then iteration stops and function returns the iterable with the first element that is False

```

```

irst results

return_if_first_odd = itertools.dropwhile(lambda item: item%2!=0, [1, 3, 5,
4, 7, 6, 8, 9])
print(list(return_if_first_odd))

return_if_first_even = itertools.dropwhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are False to the predicate until
meets True, then iteration stops and function returns the iterable with the
first True result and the rest after it

```

→

[1, 3, 5]
[2, 2, 4, 8]
[4, 7, 6, 8, 9]
[1, 2, 4]

- ▼ `takewhile(predicate, iterable)` - возвращает элементы из итерируемого объекта до тех пор, пока предикат имеет значение True (finite iterator)

```

import itertools

greater_three = itertools.takewhile(lambda item: item<3, [1, 2, 3, 4])
print(list(greater_three))

```

→

[1, 2]

```

import itertools

return_if_first_odd = itertools.takewhile(lambda item: item%2!=0, [1, 3, 5,
4, 7])
print(list(return_if_first_odd))

return_if_first_even = itertools.takewhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are True to the predicate until
meets False, then iteration stops and function returns the iterable with the
first results

```

```

return_if_first_odd = itertools.dropwhile(lambda item: item%2!=0, [1, 3, 5,
4, 7, 6, 8, 9])
print(list(return_if_first_odd))

return_if_first_even = itertools.dropwhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are False to the predicate until
meets True, then iteration stops and function returns the iterable with the
first True result and the rest after it

```

→

- [1, 3, 5]
- [2, 2, 4, 8]
- [4, 7, 6, 8, 9]
- [1, 2, 4]

▼ `filterfalse()` - фильтрует элементы итерируемого объекта, возвращая только те, для которых предикат имеет значение `False`. Если предикат равен `None`, он возвращает элементы, которые стоят в значении `False`. (finite iterator)

```

import itertools

filter_odd = itertools.filterfalse(lambda item: item%2==0, [1, 2, 3, 5, 4, 7,
6, 8])
print(list(filter_odd))

filter_none = itertools.filterfalse(None, [0, 1, 2, 3, 4])
print(list(filter_none))

```

→

- [1, 3, 5, 7]
- [0]

▼ `zip_longest(iterables, fillvalue=None)` - агрегирует элементы из каждого итерируемого объекта. Если итераторы имеют неравномерную длину, то на место пропущенных значений ставится *fillvalue*. Итерация будет продолжаться до тех пор, пока не закончится самый длинный итерируемый объект. (finite iterator)

```

import itertools

zipped = itertools.zip_longest(["Gucci", "Chanel", "Prada"], ["in stock", "not in stock", "in stock", "not in stock", "in stock", "not in stock", "in stock", "not in stock"])
print(list(zipped))

zipped1 = itertools.zip_longest(["cat", "dog", "mouse", "hamster", "bunny"],
                               ["fed", "fed"], fillvalue="mb need to feed")
print(list(zipped1))

```

→

[('Gucci', 'in stock'), ('Chanel', 'not in stock'), ('Prada', 'in stock'), (None, 'not in stock'), (None, 'in stock'), (None, 'not in stock'), (None, 'in stock'), (None, 'not in stock')]
[('cat', 'fed'), ('dog', 'fed'), ('mouse', 'mb need to feed'), ('hamster', 'mb need to feed'), ('bunny', 'mb need to feed')]

- ▼ `starmap()` - вычисляет функцию, получая аргументы из итерируемого объекта.
 Используется вместо `map()`, когда параметры аргумента уже сгруппированы в кортежи в одном итерируемом объекте (данные были предварительно сжаты).
(finite iterator)

```

import itertools

a2=itertools.starmap(lambda x:x**2,[(1, ),(2, ),(3, )])
print (list(a2))

sq = itertools.starmap(pow, [(1, 2), (2, 2), (3, 2)])
print(list(sq))

```

→

[1, 4, 9]
[1, 4, 9]

- ▼ `islice()` - возвращает выбранные элементы из итерируемого объекта.
 Default `start` - 0, `step` - 1, `stop` - до конца итерируемого объекта. Если его нет, итератор остановится на определенной позиции. Значения `start`, `stop` и `step > 0`.
(finite iterator)

```
import itertools

s1 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], None)
print(list(s1))
s2 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 5)
print(list(s2))
s3 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 3, None, 2)
print(list(s3))
s3 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 2, 7, 2)
print(list(s3))
```

→

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5]
[4, 6, 8]
[3, 5, 7]
```

- ▼ `tee(iterable, n=2)` - возвращает n независимых итераторов из одного итерируемого объекта. (finite iterator)

```
import itertools

l1 = [1, 2, 3, 4, 5, 5, 6]
l2 = itertools.tee(l1, 2)
for i in l2:
    print(list(i))
```

→

```
[1, 2, 3, 4, 5, 5, 6]
[1, 2, 3, 4, 5, 5, 6]
```

- ▼ `groupby(iterable, key=None)` - возвращает последовательно ключи и группы из итерируемого объекта (finite iterator)

key – это функция, вычисляющая значение ключа для каждого элемента по умолчанию. Если ключ не указан или в значении `None`, то по умолчанию ключ является функцией идентификации, которая возвращает элемент без изменений.

```

import itertools

l1 = [("size", "XS"), ("size", "S"), ("size", "M"), ("quantity", 120), ("quantity", 134), ("size", "L")]
l2 = itertools.groupby(l1, key=lambda x:x[0])
for key, group in l2:
    result = {key: list(group)}
    print(result)
print()
l3 = [("gender", "female"), ("gender", "male"), ("age", 18), ("age", 21), ("age", 34)]
l4 = itertools.groupby(l3)
for key, group in l4:
    result = {key: list(group)}
    print(result)

```

→

```

{'size': [('size', 'XS'), ('size', 'S'), ('size', 'M')]}

{'quantity': [('quantity', 120), ('quantity', 134)]}

{'size': [('size', 'L')]}

{('gender', 'female'): [('gender', 'female')]}

{('gender', 'male'): [('gender', 'male')]}

{('age', 18): [('age', 18)]}

{('age', 21): [('age', 21)]}

{('age', 34): [('age', 34)]}

```

- ▼ `product(iterables, repeat)` - декартово произведение итерируемых объектов, подаваемых на вход.(combinatory iterator)

Декартово произведение - произведение множества X и множества Y – это множество, содержащее все упорядоченные пары (x, y) , в которых x принадлежит множеству X , а y принадлежит множеству Y .

Чтобы вычислить произведение итерируемого объекта умноженного самого на себя, нужно указать количество повторений с помощью опционального аргумента с ключевым словом `repeat`. Например, `product(A, repeat=4)` – тоже самое, что и `product(A, A, A, A)`.

```

import itertools

```

```

#Only one iterable is given
l1=itertools.product("ABCD")
print (list(l1))

#two iterables are given
l2=itertools.product("ABC", [1,2])
print (list(l2))

#one iterable and repeat is mentioned.
l3=itertools.product("xy", repeat=2)
print (list(l3))

l4=itertools.product("aa", repeat=2)
print (list(l4))

#More than two iterables is mentioned
l5=itertools.product([1,2], [3,4], [5,6])
print (list(l5))

```

→

```

[('A',), ('B',), ('C',), ('D',)]
[('A', 1), ('A', 2), ('B', 1), ('B', 2), ('C', 1), ('C', 2)]
[('x', 'x'), ('x', 'y'), ('y', 'x'), ('y', 'y')]
[('a', 'a'), ('a', 'a'), ('a', 'a'), ('a', 'a')]
[(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)]

```

- ▼ permutations() - возвращает последовательные **r** перестановок элементов в итерируемом объекте. (combinatory iterator)

Если **r** нет или **None**, то **r** = длине итерируемого объекта и генерирует все возможные полноценные перестановки. Кортежи перестановок выдаются в лексикографическом порядке в соответствии с порядком итерации входных данных. Т. о., если входные данные итерируемого объекта отсортированы, то комбинация кортежей будет выдаваться в отсортированном порядке.

Элементы рассматриваются как уникальные в зависимости от их позиции, а не от их значения. Т. о., если входные элементы уникальны, то в каждой перестановке не будет повторяющихся значений.

```

import itertools

l1=itertools.permutations("ABC")
print (list(l1))

l2=itertools.permutations([3,2,1])
print (list(l2))

#elements are treated as unique based on their position and not by their value.
l3=itertools.permutations([1,1])
print (list(l3))

l4=itertools.permutations(["ABC"])
print (list(l4))

#r value is mentioned as 2. It will return all different permutations in 2 values.
l5=itertools.permutations([1,2,3,4],2)
print (list(l5))

l4=itertools.product("aa", repeat=2)
print (list(l4))

#More than two iterables is mentioned
l5=itertools.product([1,2],[3,4],[5,6])
print (list(l5))

```

→

```

[('A', 'B', 'C'), ('A', 'C', 'B'), ('B', 'A', 'C'), ('B', 'C', 'A'), ('C', 'A', 'B'), ('C', 'B', 'A')]
[(3, 2, 1), (3, 1, 2), (2, 3, 1), (2, 1, 3), (1, 3, 2), (1, 2, 3)]
[(1, 1), (1, 1)]
[('ABC',)]
[(1, 2), (1, 3), (1, 4), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (3, 4), (4, 1), (4, 2), (4, 3)]
[('a', 'a'), ('a', 'a'), ('a', 'a'), ('a', 'a')]
[(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)]

```

▼ **combinations()** - возвращает подпоследовательности длины **r** из элементов итерируемого объекта, подаваемого на вход. (combinatory iterator)

Комбинация кортежей генерируется в лексикографическом порядке в соответствии с порядком элементов итерируемого объекта на входе. Таким образом, если входной итерируемый объект отсортирован, то комбинация кортежей будет генерироваться в отсортированном порядке.

Лексикографический порядок – способ упорядочивания слов в алфавитном порядке.

Элементы рассматриваются как уникальные в зависимости от их позиции, а не значения. Таким образом, если выходные элементы уникальны, то в каждой комбинации не будет повторяющихся значений.

```
import itertools

l5=itertools.combinations([1,2,3,4],2)
print (list(l5))
```

→

`[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]`

- ▼ `combinations_with_replacement()` - возвращает подпоследовательности длины `r` из элементов итерируемого объекта, подаваемого на вход, при этом отдельные элементы могут повторяться больше одного раза.

```
import itertools
l1=itertools.combinations("ABC",2)
print (list(l1))
l1=itertools.combinations_with_replacement("ABC",2)
print (list(l1))

l2=itertools.combinations([3,2,1],3)
print (list(l2))
l2=itertools.combinations_with_replacement([3,2,1],3)
print(list(l2))

#elements are treated as unique based on their position and not by their value.
l3=itertools.combinations([1,1],2)
print (list(l3))
l3=itertools.combinations_with_replacement([1,1],2)
print (list(l3))
```

```
#since list contains only one element, given r value is 2. So it returns empty list.  
l4=itertools.combinations(["ABC"],2)  
print (list(l4))  
#In combinations_with_replacement,it allows repeated element.  
l4=itertools.combinations_with_replacement(["ABC"],2)  
print (list(l4))
```

→

```
[('A', 'B'), ('A', 'C'), ('B', 'C')]  
[('A', 'A'), ('A', 'B'), ('A', 'C'), ('B', 'B'), ('B', 'C'), ('C', 'C')]  
[(3, 2, 1)]  
[(3, 3, 3), (3, 3, 2), (3, 3, 1), (3, 2, 2), (3, 2, 1), (3, 1, 1), (2, 2, 2), (2, 2, 1), (2, 1, 1),  
(1, 1, 1)]  
[(1, 1)]  
[(1, 1), (1, 1), (1, 1)]  
[]  
[('ABC', 'ABC')]
```

```
#r value is not mentioned. It will raise TypeError  
l5=itertools.combinations([1,2,3,4])  
print (list(l5))#Output:TypeError: combinations() missing required argument  
'r' (pos 2)  
l5=itertools.combinations_with_replacement([1,2,3,4])  
print (list(l5))#Output:TypeError: combinations_with_replacement() missing required argument 'r' (pos 2)
```

▼ dir()

- ▼ There is a built-in function to list all the function names (or variable names) in a module
 - the `dir()` function.

```
import platform  
  
x = dir(platform)  
print(x)
```

→

```
['_Processor', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES',  
'builtins', 'cached', 'copyright', 'doc', 'file', 'loader', 'name', 'package', 'spec', 'version',  
'_comparable_version', '_component_re', '_default_architecture', '_follow_symlinks',  
'_get_machine_win32', '_ironpython26_sys_version_parser',  
'_ironpython_sys_version_parser', '_java_getprop', '_libc_search', '_mac_ver_xml',  
'_node', '_norm_version', '_platform', '_platform_cache', '_pypy_sys_version_parser',  
'_sys_version', '_sys_version_cache', '_sys_version_parser', '_syscmd_file',  
'_syscmd_ver', '_uname_cache', '_unknown_as_blank', '_ver_output', '_ver_stages',  
'architecture', 'collections', 'functools', 'itertools', 'java_ver', 'libc_ver', 'mac_ver',  
'machine', 'node', 'os', 'platform', 'processor', 'python_branch', 'python_build',  
'python_compiler', 'python_implementation', 'python_revision', 'python_version',  
'python_version_tuple', 're', 'release', 'subprocess', 'sys', 'system', 'system_alias', 'uname',  
'uname_result', 'version', 'win32_edition', 'win32_is_iot', 'win32_ver']
```

▼ Import from a module →

You can choose to import only parts from a module, by using the `from` keyword (from sm import sm_person).

When importing using the `from` keyword, do not use the module name when referring to elements in the module. Example: `person1["age"]`, not `mymodule.person1["age"]`

▼ Custom modules

▼ Attributes

▼ `__name__`

When Python is running in interactive mode or as a script, its `__name__` is `_main_`. It is the name of scope in which top level is being executed. However, for imported module this attribute is set to name of the Python script. (excluding the extension .py).

▼ `__doc__`

This attribute returns the docstring of module. Just as in function, Documentation string (docstring) is a string literal in first line written in module code.

▼ `__file__`

This attribute will returns the name and path of the module file.

▼ `__dict__`

This attribute returns a dictionary object of all attributes, functions and other definitions and their respective values.

▼ Module search

How does Python interpreter find a module when a script requests it to be imported? Python follows following method to locate a module:

- Whether it is present in current working directory?
- If not found there, directories in PYTHONPATH environment variable are searched.
- Otherwise, it searches the installation dependent default directory.

▼ Reloading a module

Python loads any module only once even if import statement is issued repeatedly.

```
"docstring of messages module"
def welcome(name):
    print ("Hi {}. Welcome to Python Tutorial".format(name))
    return
def bye(name):
    print ("Good Bye {}. See you again".format(name))
welcome("world")
bye("world")
```

Let us import messages module from Python prompt. The calls to functions in the code also get executed. However, repeating import statement doesn't execute them again.

```
>>> import messages
Hi world. Welcome to Python Tutorial
Good Bye world. See you again
>>> import messages
>>>
```

Now suppose we need to modify the SayHello() function before executing it again. Updated function is :

```
"docstring of messages module"
def welcome(name):
    print ("Hi {}. Welcome to Python Tutorial".format(name))
    return
def bye(name):
```

```
    print ("Good Bye {}. See you again".format(name))
welcome("Guest")
bye("Guest")
```

This change will be effected only if current interpreter session is closed and re-launched. If such changes are to be done frequently, to close and restart Python is cumbersome. In order to call load the module without terminating interpreter session, use reload() function from imp module.

```
>>> import imp
>>> imp.reload(messages)
Hi Guest. Welcome to Python Tutorial
Good Bye Guest. See you again
<module 'messages' from 'C:\\python37\\messages.py'>
```

Decorators

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-decorators>

CGI in Python

Common Getaway Interface - стандарт для внешних шлюзовых программ для взаимодействия с информационными серверами (пр.: HTTP-серверы). Это набор соглашений, к-й должен соблюдаться web-серверами при выполнении ими различных web-приложений.

Переменные окружения

- ▶ QUERY_STRING – храниться значения
- ▶ REQUEST_METHOD – храниться метод передачи данных get или post
- ▶ CONTENT_TYPE – хранит тип передачи данных
Для get обычно используется "application/x-www-form-urlencoded"
Для post обычно используется "multipart/form-data"
- ▶ HTTP_HOST – хранит имя хоста с портом
- ▶ SERVER_NAME - хранит имя хоста
- ▶ HTTP_USER_AGENT – хранит сведения о клиенте "Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0"
- ▶ HTTP_REFERER – формируется автоматически браузером и хранит url страницы с которой пришёл запрос

▶ CONTENT_LENGTH – хранит строку, являющуюся десятичным представлением длины данных в байтах. Присутствует только в методе POST, в GET отсутствует

▶ HTTP_COOKIE – хранит все cookies в URL-кодировке

▶ HTTP_ACCEPT – хранит перечисления типов данных документа который принимает браузер например: "text/html, text/plain, image/gif, image/jpeg", но все новые браузеры передают "*" – это значит что принимают все типы.

Подробнее: <https://andreyex.ru/yazyk-programmirovaniya-python/uchebnik-po-python-3/python-3-programmirovaniye-v-python-cgi/>

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-cgi>

Django

Документация: <https://docs.djangoproject.com/en/4.0/ref/contrib/messages/>

▼ Методика MTV (или MVC)

Описывает общий дизайн БД ориентированных веб-приложений Django.

Django поощряет свободное связывание и строгое разделение частей приложения. Поэтому можно легко вносить изменения в одну конкретную часть приложения без ущерба для остальных частей. Так, мы используем тот же принцип разделения, как если бы отделяли бизнес-логику от логики отображения с помощью шаблонной системы.

Эти три вещи вместе — логика доступа к данным, бизнес-логика и логика отображения — составляют концепцию, которую называют шаблоном *Модель-Представление-Управление*

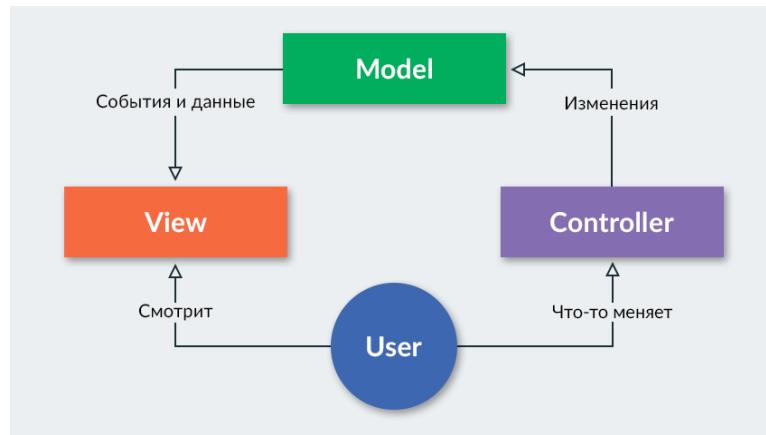
(*Model-View-Controller*, MVC) архитектуры программного обеспечения. В этой концепции термин «Модель» относится к логике доступа к данным; термин «Представление» относится к той части системы, которая определяет, что показать и как; а термин «Управление» относится к той части системы, которая определяет какое представление надо использовать, в зависимости от пользовательского ввода, по необходимости получая доступ к модели.

Django следует модели MVC достаточно близко, т.е., может быть назван MVC совместимой средой разработки. Вот примерно как M, V и C используются в Django:

- *M*, доступ к данным, обрабатывается слоем работы с базой данных, который описан в этой главе.
- *V*, эта часть, которая определяет какие данные получать и как их отображать, обрабатывается представлениями и шаблонами.
- *C*, эта часть, которая выбирает представление в зависимости от пользовательского ввода, обрабатывается самой средой разработки, следуя созданной вами схемой URL, и вызывает соответствующую функцию Python для указанного URL.

Так как «С» обрабатывается средой разработки и всё интересное в Django происходит в моделях, шаблонах и представлениях, на Django ссылаются как на *MTV-ориентированную среду разработки*. В MTV-подходе к разработке:

- *M* определено для «Модели» (Model), слоя доступа к данным. Этот слой знает всё о данных: как получить к ним доступ, как проверить их, как с ними работать и как данные связаны между собой.
- *T* определено для «Шаблона» (Template), слоя представления данных. Этот слой принимает решения относительно представления данных: как и что должно отображаться на странице или в другом типе документа.
- *V* определено для «Представления» (View), слоя бизнес-логики. Этот слой содержит логику, как получать доступ к моделям и применять соответствующий шаблон. Вы можете рассматривать его как мост между моделями и шаблонами.



REST Framework

CORS headers

▼ Протоколы прикладного уровня

- [9P](#)
- [BitTorrent](#)
- [BOOTP](#)
- [DNS](#)
- [FTP](#)
- [HTTP](#)

- NFS
- POP, POP3
- IMAP
- RTP
- SMTP
- SNMP
- SPDY
- Telnet
- SSH
- X.400
- X.500
- RDP
- Matrix
- SFTP

Компьютерные сети:

- по типу коммутации: комм. каналов (по каналу, пр.: тел. сети) и комм. пакетов (каждый пакет по своему пути, пр.: комп. сети)
- по технологии передача: широковещательные сети (сразу всем устр.) и “точка-точка”(от одного другому, мб посредством третьего и т.д.)
- протяженность
(персональная → локальная → муниципальная → глобальная → объединение сетей)

Топология КС - способ объединения комп. в сеть (еще наз. “граф”)

Вершины графа - узлы сети (комп. и сет. устр), ребра - связи между узлами (физ. и инф.)

- полносвязная Т - каждый соед. с каждым
- ячеистая Т - как полносвязная, но без некоторых соед.
- звезда - все подсоединенены к центральному устр-ву (коммутатор, ви-фи, маршрутизатор и т.д.), через к-е и осущ. передача данных

- кольцо - передача по кругу через соседей
- дерево - комп. и ЦУ обр-т дерево (ЦУ → др. ЦУ → комп.)
- общая шина - все комп. подкл к общ. сети передачи (пр. медный кабель), как к каналу
- смешанная - (на практике) - пр.: ЦУ в кольце, к ним подкл-ы комп-ы по звезде, а ост-е - по дереву через доп. ЦУ.

Физич. Т - соед. устр-в в сети. Логич. Т - правила расп-я сигналов в сети. Могут различаться: Ethernet - физ-ки звезда, лог-ки шина. Коммутир. Ethernet - ф-ки звезда, л-ки полносвяз. WiFi - физ. - нет, лог. - шина.

Стандарты сетей:

- Эталонная модель взаимодействия открытых сетей (принята OSI, м-н орг. по стандартизации)
- Технологии передачи данных (институт инж. по эл-ке и электротехнике, IEEE)
- Протоколы интернет (прин. Совер по арх. интернета, IAB)
- стандарты Web (консорциум W3C)

Для решения сложностей в организации масштабируемой КС был создан метод декомпозиции задач “уровни”. У кажд. ур. своя задача (или набор связ. задач).

Сервис - опис-т фу-ии ур-я.	Интерфейс - набор примитив. опер., к-е ниж. ур. предост. верх.	Протокол - прав. и соглаш. для связи ур. N 1 комп. с ур. N 2 комп.
--------------------------------	--	---

Арх. сети - набор ур. и протоколов (интерфейсы не входят). Стек протоколов собран иерархически.

Модель OSI

Хорошая теор. детализация, но на практ не прим-ся. Open Systems Interconnection, Модель взаимодействия открытых систем. Октр. сист. - в соотв. с открытыми спецификациями.

7. Ур. приложения (сообщение) - он же прикладной - включает все сервисы, к-ми может пользоваться пользователь (гипертекстовые web-страницы, соц. сети, видео и аудио связь, эл. почта, доступ к разделяемым файлам и т.д.).

Здесь располагаются сетевые службы, позволяющие пользователю взаимодействовать с машиной: Telnet, LPD, TFTP, NFS, DNS, DHCP, SNMP, X Window.

- Протокол HTTP (Hyper-Text Transfer Protocol) - протокол передачи гипертекста, основа www.

URL (Uniform Resource Locator) - уникальное положение ресурса. HTTP работает в режиме запрос-ответ.

Постоянное TCP соединение служит для того, чтобы по протоколу HTTP передавалось не по 1 файлу за соединение, а сразу все. (keep-alive или persistent connection).

В HTTP 1.0 нужно добавить строчку Connection: keep-alive, а в HTTP 1.1 все соединения по умолчанию постоянные (чтобы разорвать, нужно добавить заголовок Connection: close)

Минусы: клиент открыл соединение и не использует его - ресурсы недоступны другим, плохо для высоконагруженных серверов. Решение: автоотключение через таймаут 5-15 сек., или еще конвейреная обработка (pipelining) (несколько запросов → несколько ответов, недостаток - сохранение порядка, решена в HTTP2, где есть нумерация запросов), или еще вариант - несколько HTTP соединений (каждое м.б. постоянным или использовать конвейерную обработку). Последний вариант сейчас чаще всего используется.

Кэширование сокращает время загрузки страницы, но требует место на лок. диске комп-а. Может кэшировать отдельные ресурсы, к-е редко меняются.

Заголовок Expires помогает понять, можно ли взять страницу из кэша или ее нужно обновить. Если в браузере такого нет, используются эвристики (Last-Modified). Другой способ (нивелирующий ошибки предыдущих) - GET Conditional (при наличии Last-Modified): были ли изменения? В совр. версиях HTTP также используется ETag (entity tag) в запросах GET с условием.

Заголовок Cache-Control служит для управления кэшем.

Альтернатива хранению на лок. диске комп-а - прокси-сервер с разделяемым кэшем.

Есть еще обратный прокси, к-й устанавливается не со стороны клиента, а со стороны вэб-серверов.

Порядок следования HTTP сообщение:

- HTTP-метод
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (<http://>), домена (здесь developer.mozilla.org), или TCP порта (здесь 80)
- Версию HTTP-протокола

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера
- Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс

Методы HTTP

GET – запрос Web-страницы
POST – передача данных на Web-сервер
HEAD – запрос заголовка страницы
PUT – помещение страницы на Web-сервер
DELETE – удаление страницы с Web-сервера
TRACE – трассировка страницы
OPTIONS – запрос поддерживаемых методов HTTP для ресурса
CONNECT – подключение к Web-серверу через прокси

Статусы HTTP

1XX – информация
2XX – успешное выполнение (200 OK)
3XX – перенаправление (301 – постоянное перемещение, 307 – временное перенаправление)
4XX – Ошибка на стороне клиента (403 – доступ запрещен, 404 – страница не найдена)
5XX – Ошибка сервера (500 – внутренняя ошибка сервера)

Структура пакета HTTP

→ Запрос/статус ответа

- GET /courses/networks
- 200 OK

Заголовки (не обязательно)

- Host: www.asozykin.ru (обязательно в HTTP 1.1)
- Content-Type: text/html; charset=UTF-8
- Content-Length: 5161

Тело сообщения (не обязательно)

- Страница HTML
- Параметры, введенные пользователем

Пример запроса HTTP

Подключение по TCP к серверу www.asozykin.ru,
порт 80

```
-----  
GET /courses/networks HTTP/1.1  
Host: www.asozykin.ru
```

Пример ответа HTTP

```
HTTP/1.1 200 OK  
Server: nginx  
Content-Type: text/html; charset=UTF-8  
Content-Length: 5161
```

```
<html lang="ru-RU">  
<head>  
...  
</html>
```

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь `developer.mozilla.org`), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь `developer.mozilla.org`), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь developer.mozilla.org), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

- Протокол SMTP (Simple Mail Transfer Protocol) - простой протокол передачи почты (посл. версия - ESMTP, extended). Схему использования протокола при передаче почты - см. ниже.

Теоретически SMTP может использовать любой транспортный протокол (TCP, UDP, др.). Порты: 25 для передачи почты между серверами, 587 для приема почты от клиентов. На практике: протокол TCP, порт 25.

Протокол SMTP входит в конверт письма, а заголовки и содержимое письма включают RFC 2822.

Протокол работает в формате текста в режиме запрос-ответ.

Минусы протокола: незащищенность данных, спам.

Команды SMTP

Команда	Назначение	Пример
HELO	Установка соединения	HELO example.com
MAIL	Адрес отправителя	MAIL FROM: sender@example.com
RCPT	Адрес получателя	RCPT TO: recipient@mail.ru
DATA	Передача письма	DATA
QUIT	Выход	QUIT

Ответы SMTP

Код	Назначение	Пример
220	Подключение к серверу успешно	220 smtp.example.com ESMTP Postfix
250	Успешное выполнение предыдущей команды	250 Hello example.com 250 Ok
354	Начало передачи письма	354 End data with <CR><LF>.<CR><LF>
502	Команда не реализована	502 5.5.2 Error: command not recognized
503	Неправильная последовательность команд	503 5.5.1 Error: need MAIL command
221	Закрытие соединения	221 2.0.0 So long, and thanks for all the fish

Заголовки письма

Заголовок	Назначение
From:	Отправитель (имя и адрес)
To:	Получатель
CC:	Получатель копии письма
BCC:	Получатель копии, адрес которого не должен быть показан
Reply-To:	Адрес для ответа
Subject:	Тема письма
Date:	Дата отправки письма

- Протокол IMAP (Internet Message Access Protocol) - протокол доступа к эл. почте. Письма хранятся на почтовом сервере. Клиенты подключаются к серверу и загружают письма только после запроса пользователя. Сервер может выполнять сложные операции с письмами.

Преимущества: одновременно могут работать несколько клиентов, все клиенты видят одно и то же состояние почтового ящика.

Недостатки: более сложный протокол, ограниченное место для почтового ящика на сервере.

IMAP использует TCP, порт 143.

IMAP позволяет использовать несколько почтовых ящиков и папок (хранятся на сервере, можно перемещать и образовывать иерархию). Также есть флаги.

Состояния сеанса: клиент не аутентифицирован → клиент аутентифицирован → папка выбрана → выход.

Работает в текстовом режиме, взаимодействие запрос-ответ. Позволяет выполнять несколько команд одновременно (есть теги команд).

Статусы: OK, NO (ошибка), BAD (неправильная команда).

- Протокол POP3 (Post Office Protocol) - протокол почтового отделения. Работает по принципу загрузить и удалить. Ящик на сервере - временное хранилище, сообщения переписываются на почтовый клиент, после чего удаляются с сервера.

Плюсы: простота, письма доступны при отсутствии подключения к сети. Минусы: только один клиент, единое хранилище писем (нет папок, фильтров и т.д.).

Состояния сеанса: авторизация, транзакция, обновление.

Протокол работает в текстовом режиме, взаимодействие запрос-ответ.

Ответы протокола: + OK или - ERR.

Команды POP3

Команда	Назначение	Пример
USER	Указать имя пользователя	USER asozykin
PASS	Указать пароль	PASS 1234qwer
STAT	Количество писем на сервере	STAT
LIST	Передача информации о сообщениях	LIST 2
RETR	Передать сообщение на клиент	RETR 1
TOP	Передать на клиент заголовок сообщения	TOP 2 10
DELE	Пометить сообщение на удаление	DELE 1
QUIT	Закрытие транзакции, удаление сообщений и отключение	QUIT

- **Протокол DNS** (Domain Name System) - система доменных имен. Позволяет использовать вместо неудобных IP понятные для пользователя доменные имена. Также позволяет менять сетевую инфраструктуру (при переходе на другой IP домен менять не надо). Один домен может обслуживать несколько серверов. Узнать IP по домену можно с помощью утилиты nslookup. Для Linux это утилиты host и dig.

Особенности DNS:

- распределенная (децентрализованная) система (нет единого сервера, на котором описываются имена хостов)
- делегирование ответственности (пространство имен разделено на отдельные части - домены, за каждый домен твердит отдельная организация)
- надежность (дублирование серверов DNS)

Структура корневого домена: www(имя компьютера).wiki(домен второго уровня).com(домен верхнего уровня).(корневой домен)

Важная особенность - делегирование ответственности за хранение данных между доменами разных уровней.

Распределением имен занимаются регистраторы.

Режимы работы DNS:

1) Итеративный:

- если сервер отвечает за данную доменную зону - он возвращает ответ
- если нет - возвращает адрес DNS-сервера, у которого есть более точная информация

2) Рекурсивный:

- сервер сам выполняет запросы к другим серверам DNS, чтобы найти нужный адрес
Сервер разрешения имен DNS предоставляет провайдером/организацией, комп.
получает адрес локального DNS по DHCP.

Как только адрес найден, DNS сервер записывает его в кэш.

Типы ответа DNS:

- 1) авторитетный: ответ от сервера, обслуживающего доменную зону; получен из файлов на диске сервера
- 2) неавторитетный: ответ от сервера, не обслуживающего доменную зону; получен из кэша, данные могли устареть.

Работает по модели клиент-сервер. Взаимодействие идет в режиме запрос-ответ. DNS использует протокол UDP, номер порта 53.

Что еще может DNS:

- определять для доменного имени адреса IPv4 и IPv6
- задавать несколько доменных имен для одного IP-адреса
- находить адрес почтового сервера для домена
- определять IP-адрес и порт некоторых сетевых сервисов
- задавать адрес DNS-серверов для доменной зоны
- определять по IP-адресу доменное имя

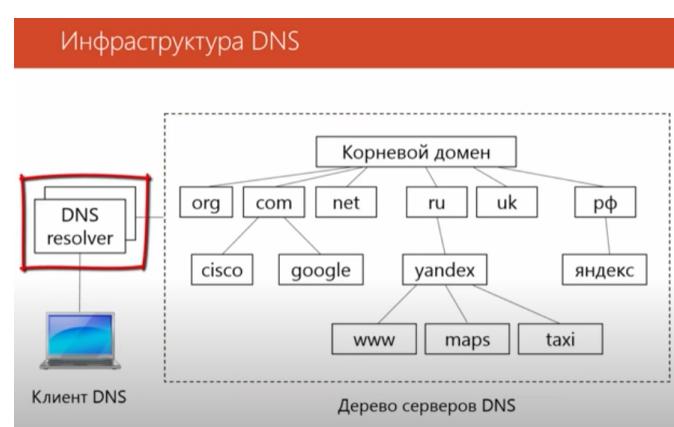
Запись MX нужна для отправки почты на почтовый сервер.

Чтобы присвоить несколько имен: псевдоним или несколько A записей.

Для некоторых сервисов можно задавать не только IP-адреса, но и номера портов. Для этого используется SRV запись.

Запись типа NS (name server) указывает адреса DNS серверов, отвечающих за зону.

Записи типа PTR используются реверсивной зоной для определения имени по IP.



Формат пакета DNS



- Протокол FTP (File Transfer Protocol) - протокол передачи файлов.

Работает по модели клиент-сервер. На сервере есть структура файлов. Клиент по протоколу подключается к серверу и работает с файлами.

Как и HTTP использует URL (`ftp://ftp-server.ru/path/file.format`). Использует два соединения: управляющее и для передачи данных.

Использует TCP, порт 21. Соединение данных: в активном режиме - порт сервера 20, в пассивном - порты больше 1024 (исп. если между клиентом и сервером есть интерфейс, напр. экран).

Есть аутентификация и режим анонимса.

Недостатки: проблемы с NAT (решает пассивный режим), низкая безопасность.

Замена: протоколы на основе SSH (SFTP, SCP).

Протокол FTP

Команда	Назначение
USER	Указать имя пользователя
PASS	Указать пароль
LIST	Просмотр содержимого каталога
CWD	Смена текущего каталога
RETR	Передать файл с сервера на клиент
STOR	Передать файл с клиента на сервер
TYPE	Установить режим передачи
DELE	Удалить файл
MKD	Создать каталог
RMD	Удалить каталог
PASV	Использовать пассивный режим
QUIT	Выход и разрыв соединения

6. Ур. представления (сообщение) - обеспечивает согласование синтаксиса и семантики передаваемых данных (форматы представления символов, форматы чисел). Шифрование и дешифрование. Пр.: Transport Layer Security (TLS), Secure Sockets Layer (SSL). Тут происходит кодирование и сжатие (пр.: JPEG, GIF).

Для защиты передаваемых по сети данных часто используется шифрование:

- Secure Socket Layer (SSL)
- Transport Layer Security (TLS)

Протоколы, которые используют SSL/TSL:

- HTTPS, порт 443
- IMAPS, порт 993
- SMTPS, порт 465
- FTPS

5. Сеансовый (сообщение) - позволяет устанавливать сеансы связи. Управляет диалогом (очередность передачи сообщений). Управляет маркерами (предотвращение одновременного выполнения критичной операции). Синхронизация (метки в сообщениях для возобновления передачи в случае сбоя).

Сеанс (сессия) - набор связанных между собой сетевых взаимодействий, направленных на решение одной задачи.

Загрузка Web-страницы:

- загрузка текста (.html)
- загрузка стилевого файла (.css)
- загрузка изображений

Подходы к загрузке Web-страницы:

- для каждого элемента создается отдельное соединение (HTTP 1.0)
- загрузка всех элементов через одно соединение TCP (HTTP keep-alive)

Пример взаимодействия на сеансовом уровне: аудио и видео конференция, т.к. на этом уровне устанавливается, каким кодаком будет кодироваться сигнал (кодаки должны совпадать с обеих сторон).

4. **Транспортный** (сегмент, дейтаграмма) - обеспечивает передачу данных между процессами на хостах. Надежность выше сети, гарант. порядок сообщений. TCP (все в точности) и UDP (может что-то потеряться). Сквозной уровень, т.к. дотавляет сообщения от источника адресату, в то время как предыдущие уровни исп. принцип звеньев (т.ж. тр. ур. называют сетенезависимым).

У хостов есть все 7 уровней, а вот у сетевого оборудования - первые 3. Транспортный уровень может связывать хосты, минуя сетевое оборудование - сквозное соединение. Важно указать адресацию, чтобы понимать, для какого процесса предназначен пакет. Для адресации используются порты. Это число от 0 до 65535. Каждое сетевое приложение на хосте имеет свой порт. Номера портов у приложений не повторяются. Формат записи: IP:порт.

Хорошо известные порты:

- 80 - HTTP (Web)
- 25 - SMTP (email)
- 53 - DNS
- 67, 68 - DHCP
- использовать может только root/ админ

Зарегистрированные порты: 1025-49151 (рег. в IANA). Динамические порты автоматически назначаются ОС клиенту сетевым приложениям.

Так, например, если мы (клиент) откроем браузер 1 и сделаем запрос на web-сервер (daemon), то сервер увидит наш IP и порт, и отправит ответ на порт браузера 1. Если откроем браузер 2 и сделаем запрос, ответ придет уже на порт браузера 2, который автоматически присвоила ему наша ОС при открытии.

Надежность уровня:

- гарантированная доставка данных:
 - подтверждение получения
 - данные отправляются снова, если не было подтверждения доставки
- гарантированный порядок следования сообщений - нумерация сообщений

Для взаимодействия с транспортным уровнем используется интерфейс сокетов:

Интерфейс транспортного уровня TCP/IP



UDP (User Datagram Protocol) - протокол действа грам пользователя. (дейтаграмма - сообщение UDP).

Особенности:

- нет соединения, за счет этого быстрее TCP
- нет гарантии доставки данных
- нет гарантии сохранения порядка сообщений

Зачем нужен? На транспортном уровне необходимо указать порты отправителя и получателя, что и делает протокол.

По надежности: в совр. сетях ошибки редки, да и обработать их могут приложения

Область применения: клиент-сервер и короткие запросы-ответы. Пример: DNS (клиент-DNS - сервер-DNS).

TCP (Transmission Control Protocol) - протокол управления передачей.

Обеспечивает надежную передачу потока байт (reliable byte stream).

Гарантии: доставка данных и сохранение порядка следования сообщений.

В протоколе TCP поток данных делится на отдельные сегменты. Они отправляются отдельно к получателю, который в свою очередь собирает их обратно в поток байт.

Чтобы обеспечить гарантию передачи данных TCP использует подтверждение получения.

Также TCP использует метод скользящего окна, подтверждая не каждый сегмент, а совокупность.

Для гарантии сохранения порядка отправления, TCP нумерует байты (сегментами по 1024

байт с 0 байт). Если произошла ошибка при отправке подтверждения о получении, получатель видит, что у него уже есть этот фрагмент и повторно отправляет подтверждение.

Перед передачей данных TCP необходимо установить соединение (это замедляет процесс).

Задачи соединения:

- убедиться, что отправитель и получатель хотят передавать друг другу данные
- договориться о нумерации потока байт
- договориться о параметрах соединения (макс. размер сегмента и т.д.)

После завершения передачи данных соединение разрывается.

Варианты подтверждения доставки:

- остановка и ожидание (WiFi, канальный ур.): отправка данных → ожидание подтверждения получения → подтверждение получения → продолжение отправки данных и т.д. (медленно, локальный)
- скользящее окно (TCP, транспортный уровень): отправка нескольких порций данных без ожидания подтверждения получения → кумулятивное подтверждение (получил последнюю порцию данных и все предыдущие) (быстро, для больших сетей)
Размер окна - кол-во байт данных, к-е могут быть переданы без получения подтверждения.

Еще есть выборочное подтверждение (Selective Acknowledgement, SACK) - подтверждение диапазонов принятых байт, эффективно при большом размере окна, доп. поле заголовка TCP (параметр).

Установление соединения:

- запрос на соединение (SYN) + порядковый номер передаваемого байта
- подтверждение соединения + информ. о предыдущих принятых байтах + ACK с номером ожидаемого байта
- подтверждение получения подтверждения о соединении с номером предыдущего полученного байта + ACK с номером следующего к передаче байта
- соединение установлено

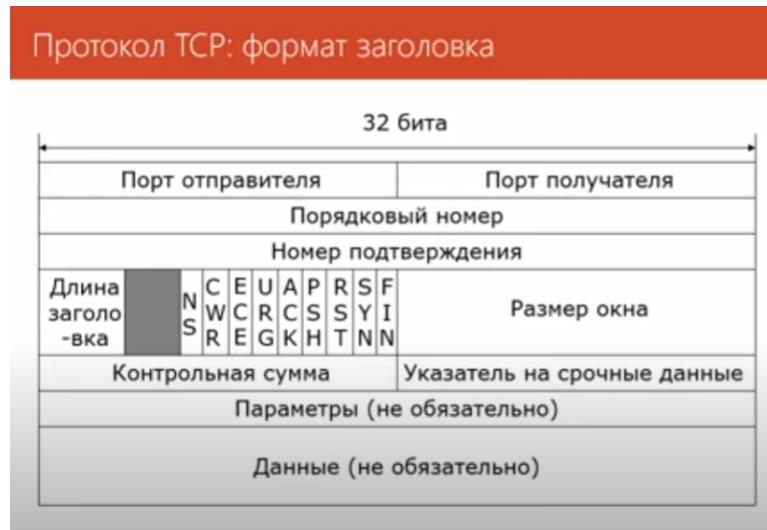
Разрыв:

- одновременное (обе стороны разорвали соединение)

- одностороннее (одна сторона прекращает передавать данные, но может принимать)

Есть варианты разрыва соединения: одностороннее (FIN) и из-за критической ситуации (RST, в обе стороны)

Порядок: FIN1 → ACK2 → (K2 закончил передачу данных) FIN2 → ACK1.



Управление потоком - справляется с проблемой “затопления”. Для этого есть поле “размер окна”.

Борьба с перегрузкой: окно перегрузки, которое рассчитывается в зависимости от нагрузки на сеть. В TCP есть AIMD (Additive increase/multiplicative decrease) - аддитивное увеличение/мультипликативное уменьшение - определяет размер окна перегрузки в зависимости от того, есть или нет перегрузки, опираясь на макс. размер сегмента. Сигнал о перегрузке - потеря сегмента или задержка сегмента или сигнал от маршрутизатора. Альтернатива - медленный старт: малый первоначальный размер окна перегрузки, при каждом подтверждении отправляется 1 сегмента, происходит экспоненциальный рост размера окна, а после сигнала о перегрузке все начинается с начала.

Интерфейс сокетов - для работы программиста с ПО на транспортном уровне (файлы).

Операции сокетов Беркли

Операция	Назначение
Socket	Создать новый сокет
Bind	Связать сокет с IP-адресом и портом
Listen	Объявить о желании принимать соединения
Accept	Принять запрос на установку соединения
Connect	Установить соединение
Send	Отправить данные по сети
Receive	Получить данные из сети
Close	Закрыть соединение

Работает по клиент-серверной модели. Сервер - программа, которая работает (слушает) на известном IP-адресе и порте и пассивно ждет запросов на соединение. Клиент - приложение, которое активно устанавливает соединение с сервером на заданом IP и порте.

NAT (Network Address Translation) - трансляция сетевых адресов (решение проблемы нехватки адресов IPv4). Это технология преобразования IP-адресов внутренней (частной) сети в IP-адреса внешней сети (Интернет).

Типы:

- статический (сколько комп=ов, столько и адресов, подх. для подключения к др. корп. сети)
- динамический (есть несколько IP для внеш. сети, которые поочередно используются комп-ами)
- один ко многим (masquerading) (один адрес на всех)

Межсетевой экран - отделяет сеть от других сетей (он же брандмауэр, firewall).

Перехватывает и проверяет пакеты.

Как обезопасить:

- флаг ACK - злоумышленник не сможет установить соединение (транспортный уровень)
- разрешить получение пакетов только от того, с кем установлено соединение (транспортный уровень)
- фильтрация на портах коммутатора по MAC-адресам (канальный уровень)

- прокси-сервер (прикладной уровень)
- фильтр содержимого (прикладной уровень)
- система обнаружения вторжений (IDS)
- система предотвращения вторжений (IPS)

Может замедлять и затруднять (а то и сделать невозможной) работу компьютера.

3. **Сетевой** (пакет)- объединяет сети, построенные на осн. разных технологий.

Обеспечивает: создание составной сети, согласование различий в сетях; адресацию (сетевые и глобальные адреса); опр. маршрута пересылки пакетов в сост. сети (маршрутизация). Оборудование: маршрутизатор. Использует IP адреса.

Объединяет сети разл. технологий (Ethernet, WiFi, 3/4/5G, MPLS). Этот уровень позволяет более удобно организовать масштабную сеть, чем с вифи и езернет.

Масштабируемость на сетевом ур.:

- агрегация адресов: работа с блоками адресов (блок адресов - сеть)
- запрет пересылки “мусорных” пакетов: т.е. тех, которые непонятно, куда отправлять
- возможность наличия неск. путей в сети: допускается неск. активных путей задача выбора лучшего пути - маршрутизация

Итого, сетевой уровень решает задачи: объединения сетей, маршрутизации, обеспечения качественного обслуживания.

Протоколы:

- **IP** - протокол передачи данных.

Передача данных: без гарантии доставки, без сохранения порядка следования сообщений. Протокол IP использует передачу данных без установки соединения.

Задачи: объединение сетей, маршрутизация, качество обслуживания.

Маршрутизация - поиск маршрута доставки пакета между сетями через транзитные узлы (маршрутизаторы).

Фрагментация - разделение пакета на несколько частей (фрагментов) для передачи по сети с маленьким MTU (max. transmission unit).

Формат заголовка IP-пакета

4 бита Номер версии	4 бита Длина заголовка	8 бит Тип сервиса	16 бит Общая длина			
		16 бит Идентификатор пакета	3 бита Флаги	13 бит Смещение фрагмента		
	8 бит Время жизни	8 бит Тип протокола	16 бит Контрольная сумма			
32 бита IP-адрес отправителя						
32 бита IP-адрес получателя						
Опции и выравнивание (не обязательно)						

- ICMP (управляющий протокол, доп.) - Internet Control Message Protocol - протокол межсетевых управляющих сообщений. Служит для управления сетью. Позволяет получать сообщения об ошибках сети (получатель недоступен, закончилось TTL - время жизни пакета, запрещено фрагментировать пакет). Служит для тестирования работы сети: ping (проверка доступности получателя) и traceroute (определение маршрута к получателю). Т.о. он нивелирует недостатки протокола IP (возможная потеря данных без дальнейшего восстановления передачи и без оповещения об ошибке). Есть утилита PING, к-я помогает обпределить наличие ошибок сети.

Типы ICMP- сообщений

Тип	Назначение сообщения
0	Эхо-ответ
3	Узел назначения недостижим
5	Перенаправления маршрута
8	Эхо-запрос
9	Сообщение о маршрутизаторе
10	Запрос сообщения о маршрутизаторе
11	Истечение времени жизни пакета
12	Проблемы с параметрами
13	Запрос отметки времени
14	Ответ отметки времени

Коды ICMP- сообщений (для типа 3)

Код	Причина
0	Сеть недостижима
1	Узел недостижим
2	Протокол недостижим
3	Порт недостижим
4	Ошибка фрагментации
5	Ошибка в маршруте источника
6	Сеть назначения неизвестна
7	Узел назначения неизвестен
8	Узел-источник изолирован
9	Административный запрет

- **ARP** (управляющий протокол, доп.) - Address Resolution Protocol - чтобы по IP определить протокол канального уровня. Протокол разрешения адресов. Связывает сетевой и канальный уровни. Позволяет определить по IP-адресу компьютера его MAC адрес.

Таблица соответствия (пр.: Linux “\$ cat /etc/ethers”): IP-MAC. В крупных сетях - ARP. Работает в режиме “широковещательный запрос-ответ компьютера, узнавшего свой IP”. Кеширование в ARP-таблицу (команда arp - a) на стороне отправителя.

Т.к. ниже сетевого уровня, пакеты ARP не проходят через маршрутизатор.

Зачем узнавать MAC-адрес? Сама технология (пр. Ethernet) не знает о том, что такое IP-адрес и для отправки информации ей нужен MAC.

- **DHCP** (управляющий протокол, доп.) - Dynamic Host Configuration Protocol - чтобы авто назначать IP комп-ам в составной сети. Протокол динамической конфигурации хостов. Модель “клиент-сервер”. Discover → Offer → Request → Ack (acknowledgement).

DHCP сервер должен быть установлен в той подсети, где находится клиент, т.к. пакеты DHCP не проходят через маршрутизатор (нет широковещания). Решение: DHCP-Relay сервер.

Устройство: маршрутизатор. Протоколы маршрутизации: BGP, OSPF, RIP, EIGRP.

2. **Канальный** (кадр, фрейм) - выделяет во входящем потоке бит отдельные сообщения, обнаруживает и корректирует ошибки. В широковещат. сетях еще обесп. физическую адресацию (указание, какому комп-у это нужно передать), а также управление доступом к среде передачи данных (в один момент вр. передает один комп.). Оборудование: коммутатор, точка доступа. Использует MAC адреса.

Передача данных: с сетевого уровня устройства 1 поступает пакет на канальный уровень устройства 1, канальный уровень присваивает пакету заголовок канального уровня и концевик, и через физический уровень передает этот кадр на канальный уровень устройства 2, который считывает заголовок и концевик, извлекает пакет сетевого уровня и передает своему сетевому уровню устройству 2.

Как канальный уровень определяет кадр (методы): указатель кол-ва бит (не на практике), вставка бит/байт (сейчас: протоколы HDLC и PPP: 1). 0(1x6)0 начало и конец кадра; 2). после 1ч5 в данных добавляется 0), устр-ва физ. уровня (преамбула Ethernet классического и избыточный код Fast Ethernet нового).

Подуровни: п/у управления логическими каналами (LLC) (передача данных, мультиплексирование, управление потоком, общий для разных технологий), п/у управления доступом к среде (MAC) (совместное использование разделяемой среды, адресация, специфичный для различных технологий, не я-ся обязательным).

Ethernet:

- на канальном и физическом ур.
- протокол STP (Spanning Tree Protocol): авто отключение дублир. соединений, связующее дерево, обесп. защиту от сбоев (широковещательного штурма при образовании кольца), надежность соед. между коммутаторами (новый - RSTP, rapid)

WiFi:

- на физ. и канал. ур, подуровни канал. ур.: п/у управл. логич. каналом (LLC) и п/у управления доступом к среде (MAC).
- типы кадров: кадры данных (передача данных), к. контроля (служебные и RTS, CTS, ACK), к. управления (реа-ия сервисов вифи, пр.: точка доступа)

1. **Физический** (бит)- передает биты единым потоком в виде сигналов по физ. каналу связи, не анализируя. Оборудование: концентратор. Здесь: Ethernet, WiFi, Bluetooth, инфракрасный порт. Сетевые устройства: концентраторы и репиторы.

Как данные переходят с канального на сетевой уровень?

- внутри одной сети:
 - есть K1 с IP1 и MAC1, K2 с IP2 и MAC2. В таблицу отправления вписываются данные отправителя, из данных получателя обычно известен только IP. По ARP узнаем его MAC. От получателя получаем ответ - такую же таблицу, только в зеркальном варианте.

- разные сети:
 - есть K1 с IP1 и MAC1, K2 с IP2 и MAC2. В таблицу отправления вписываются данные отправителя, из данных получателя обычно известен только IP. Проверяем, в одной ли сети: берем адрес сети отправителя и по ней проверяем по ней адрес получателя (используется маска подсети).
 - Т.к. в разных сетях, отправляем данные сначала на маршрутизатор, адрес которого знаем из таблицы маршрутизации. При помощи ARP узнаем MAC маршрутизатора. Маршрутизатор получает пакет и формирует свою таблицу для передачи: IP остаются, а MAC адреса меняются (отправитель - маршрутизатор с интерфейсом на стороне получателя, получатель - выявляем по ARP).
 - Получатель передает ответ - таблицу в которой IP-получатель - компьютер-отправитель, MAC-получатель - MAC сети, отправитель - данные отправителя. Получатель отправляет этот пакет на маршрутизатор.
 - Маршрутизатор снова меняет: отправитель имеет IP компьютера-получателя, а MAC - MAC сети со стороны отправителя; получатель - данные получателя пакета-подтверждения.

Итого: IP всегда остается, а MAC меняется.

Модель TCP/IP

Прим-ся на практик. (т.к. удобный стек протоколов)

4. Прикладной (сеансовый+представления+прикладной) - протоколы: HTTP, SMTP, DNS, FTP.
3. Транспортный - протоколы: TCP, UDP.
2. Интернет (сетевой) - протоколы: IP (доп.: ICMP, ARP, DHCP).
1. Сетевых интерфейсов (канальный+физический) - протоколы: Ethernet, WiFi, DSL.

Инкапсуляция - вкл-е сообщения вышестоящего ур. в сообщение нижестоящего. Т.е. по мере транспортировки данных с верхнего ур. до нижнего (от польз. интерфейса к машине). происх-т инкапсуляция, а обратно - декапсуляция (от машины к польз-ю). Сообщение = заголовок+данные+концевик.

IP адрес

Сетевой уровень.

Локальные адреса

- адреса в технологии канального уровня (пр.: MAC адрес в Ethernet, IMEI адрес в 4G)
- привязаны к конкретной технологии
- не могут быть использованы в гетерогенных сетях

Глобальные адреса

- адреса сетевого уровня (пр.: IP-адреса)
- не привязаны к технологии
- применяются при объединении сетей (Интернет)

Исп-ся для уникальной идентификации комп-ов в составной сети Интернет.

Версии протокола IP:

- IPv4: 4 байта
- IPv6: 16 байт

Сетевой уровень исп-т агрегацию адресов: масштабирование - работа не с отдельными адресами, а с подсетями. (поср-вом маршрутизаторов)

Подсеть - (IP-сеть, subnet) - множество комп-ов, у которых старшая часть IP-адреса одинаковая (октет - чать IP-адреса, отделенная точками).

Структура IP-адреса:

- номер подсети - старшие биты
- номер хоста (комп-а в сети) - младшие биты

Пример: 213.180.193.3 = 213.180.193 (номер подсети) + 0.0.0.3 (номер хоста)

Маска подсети - показывает, где в IP-адресе номер подсети, а где - хоста.

Структура маски:

- длина 32 бита
- единицы в позициях, задающих номер сети
- нули в позициях, задающих номер хоста

IP-адрес (дес.): 213.180.193.3

IP-адрес: 11010101.10110100.11000001.00000011

Маска: 11111111.11111111.11111111.00000000

Подсеть (через логическое И, т.е. AND): 11010101.10110100.11000001.00000000

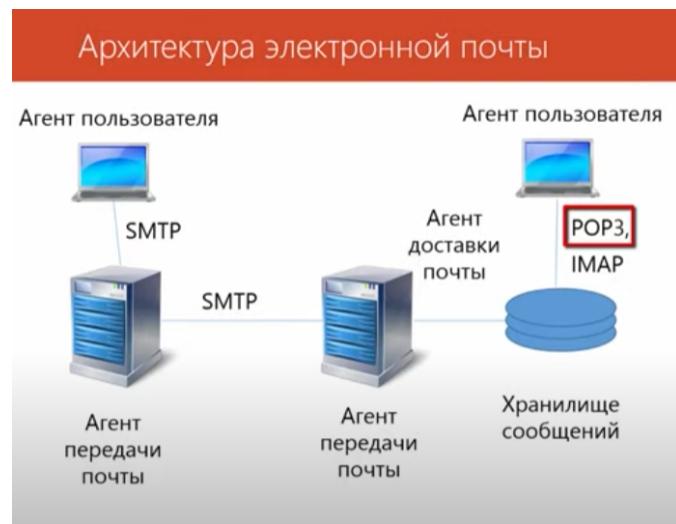
Подсеть (дес.): 213.180.193.0

Маска подсети в виде префикса (ск-ко бит - номер подсети, остаток - номер хоста):
213.180.193.3/24

Типы IP-адресов: индивидуальный (unicast), групповой (multicast) и широковещательный (broadcast, есть ограниченное и направленное, на конце адреса - кол-во всех битов).

Работа эл. почты

В почтовом адресе используется доменное имя, для его определения почтовый сервис взаимодействует с DNS, используя MX записи. Чтобы посмотреть записи MX для домена можно использовать утилиту nslookup -type=mx gmail.com



Flask

Подробнее: <https://flask.palletsprojects.com/en/2.1.x/installation/#python-version>

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Tkinter GUI

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-tkinter>

Tkinter modules

Паттерны программирования

Команды:

- ls - посмотреть, в какой сейчас директории
- cd. /name - перейти в файл

Базы данных и Python

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-database-connectivity>

Git

Шпаргалка: <https://github.com/cyberspacedk/Git-commands>

Вопрос-ответ

- **Что такое NaN ?**

Nan - Not a Number - используется для представления записей, которые не определены, а т.ж. отсутствующих значений в наборе данных.

- **Что такое var?**

▼ **Как получить все аргументы функции (включая те, что не объявлены, но все-таки были переданы)?**

```
import inspect

def pair(boy, girl):
    print(boy+" and "+girl+" are a nice couple.")

pair(boy="John", girl="Lily")
a = inspect.getfullargspec(pair)
print(a)
```

→

John and Lily are a nice couple.

```
FullArgSpec(args=['boy', 'girl'], varargs=None, varkw=None, defaults=None, kwonlyargs=[], kwonlydefaults=None, annotations={})
```

- **Что такое чистая функция?**

Чистая функция — это функция, возвращаемое значение которой зависит только от передаваемых аргументов, без побочных эффектов. Проще говоря, если вы вызываете функцию n раз с n аргументами, и функция всегда возвращает одно и тоже значение, значит, она является чистой

- **Что такое деструктуризация?**

<https://codecamp.ru/blog/python-list-destructuring-aka-packing-and-unpacking/>

- **Что такое ООП? Из чего состоит?**

<https://smartiqa.ru/courses/python/lesson-6>

Объектно-ориентированное программирование — это подход, при котором вся программа рассматривается как набор взаимодействующих друг с другом объектов. При этом нам важно знать их характеристики

Инкапсуляция — скрытие поведения объекта внутри него. Объекту «водитель» не нужно знать, что происходит в объекте «машина», чтобы она ехала. Это ключевой принцип ООП.

Наследование. Есть объекты «человек» и «водитель». У них есть явно что-то общее. Наследование позволяет выделить это общее в один объект (в данном случае более общим будет человек), а водителя — определить как человека, но с дополнительными свойствами и/или поведением. Например, у водителя есть водительские права, а у человека их может не быть. (class extends)

Полиморфизм — это переопределение поведения. Можно снова рассмотреть «человека» и «водителя», но теперь добавить «пешехода». Человек умеет как-то передвигаться, но как именно, зависит от того, водитель он или пешеход. То есть у пешехода и водителя схожее поведение, но реализованное по-разному: один перемещается ногами, другой — на машине.

Переопределение каких-то методов у различных классов. К примеру изменения метода `toString()` для конкретного класса

- **Что такое абстракция?**

Абстракция - это способ создания простой модели, которая содержит только важные свойства с точки зрения контекста приложения, из более сложной модели. Иными словами - это способ скрыть детали реализации и показать пользователям только функциональность. Абстракция игнорирует нерелевантные детали и показывает только необходимые. Важно помнить, что мы не можем создать экземпляр абстрактного класса.

Всё программное обеспечение - это абстракция, скрывающая всю тяжелую работу и бездумные детали.

Многие программные процессы повторяются снова и снова. Поэтому, на этапе декомпозиции проблемы, мы удалим дублирование, записывая какой-либо компонент (функцию, модуль, класс и т. Д.), присваивая ему имя (идентификатор) и повторно используя его столько раз, сколько нам нужно.

Процесс декомпозиции - это процесс абстракции. Успешная абстракция подразумевает, что результатом является набор независимо полезных и перекомпонованных компонентов.

- **Какие еще парадигмы знаешь?**

1) Императивный стиль — это парадигма, основанная на составлении алгоритма действий (инструкций/команд), которые изменяют состояние (информацию/данные/память) программы. Фактически, программа на этих языках — это код, который выполняется компьютером сразу, без предварительной компиляции. Из языков высокого уровня, требующих компиляции исходного кода программы в машинный код (или интерпретации), к императивным можно отнести C, C++, Java.

2) Декларативный стиль — это парадигма, при которой описывается желаемый результат, без составления детального алгоритма его получения. В пример можно привести HTML и SQL. При создании HTML мы с помощью тегов описываем, какую хотим получить страничку в браузере, а не то, как нарисовать на экране заголовок статьи, оглавление и текст. В SQL, если нам нужно посчитать количество сотрудников с фамилией «Сидоров», мы напишем `SELECT count(*) FROM employee WHERE last_name = 'Сидоров';`. Тут ничего не сказано про то, в каком файле или области памяти находятся данные по сотрудникам, как именно выбрать из них всех Сидоровых и нужно ли вообще это делать для подсчёта их количества.

Парадигмы декларативного стиля:

Логическое программирование

В целом это скорее математика, чем программирование. Его суть заключается в том, чтобы, используя математические доказательства и законы логики, решать бизнес-задачи. Чтобы использовать логическое программирование, необходимо уметь переводить любую задачу на язык математики. Логическое программирование часто используется для моделирования процессов.

Функциональное программирование

Самая известная парадигма декларативного стиля — функциональное программирование. В этой парадигме понятие функции близко к математическому понятию функции. То есть это штука, которая как-то преобразует входные данные. Особенность функции в этой парадигме в том, что она должна быть чистой, то есть должна зависеть только от аргументов и не может иметь никаких побочных эффектов. Побочный эффект — это какое-либо изменение внешней среды. Если функция меняет глобальную переменную или, например, вызывает метод внешнего объекта, она меняет внешнюю среду. Это и есть побочный эффект.

- **Как проходит инициализация класса и экземпляра класса? Какие есть свойства (поля) класса? Какие есть методы экземпляра класса?**

<https://smartiqa.ru/courses/python/lesson-6> - про классы и объекты

```
class Character:  
    def __init__(self, mana, power):  
        self.m = mana  
        self.p = power  
  
warrior = Character(100, 200)  
print(warrior.m, warrior.p)
```

- **Что геттеры и сеттеры?**
- **Какие есть статичные методы?**
- **Наследование extends**
- **Родительский конструктор: super()**
- **Работа с файлами**
Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-file-io>
- **CSV в Python**
Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-csv>

- **Родительский класс: super**
- **Какие есть способы расширения функциональности класса?**
- **Что такое промисификация?**
- **Чем отличается HTTP2 от HTTP ?**

HTTP/2 (HTTP/2.0) – это бинарный протокол, в отличии от предыдущих версий. Это значит, что данные теперь занимают меньше места и быстрее обрабатываются.

Один из основных плюсов второй версии HTTP. В предыдущей версии для одного запроса была необходима установка отдельного TCP-соединения. И чем больше было запросов, тем медленней работал браузер. Благодаря мультиплексированию браузер отправляет сразу несколько запросов через одно TCP-соединение.

Приоритизация

Первый вид приоритизации подразумевает получение потоком определенного веса, который дальше распределялся между потоками, чтобы нагрузка была равномерной. Данный тип выбора приоритета впервые был применен в протоколе SPDY.

Второй вариант является основным для http/2, а его суть заключается в том, что браузер запрашивает у сервера отдельную загрузку некоторых элементов контента, к примеру, сначала скриптов JavaScript, а затем изображений.

Если сравнивать его с прошлой версией протокола, то здесь разработчики поменяли методы распределения данных на фрагменты и их отправку от сервера к пользователю и наоборот. Новая версия протокола позволяет серверам доставлять информацию, которую клиент пока что не запросил. Это было внедрено с той целью, чтобы сервер сразу же отправлял браузеру для отображения документов дополнительные файлы и избавлял его от необходимости анализировать страницу и самостоятельно запрашивать недостающие файлы.

Еще одно отличие http 2.0 от версии 1.1 – мультиплексирование запросов и ответов для решения проблемы блокировки начала строки, присущей HTTP 1.1. Еще в новом протоколе можно сжимать HTTP заголовки и вводить приоритеты для запросов.

- **Что такое CSRF, XSS?**

CSRF (англ. cross-site request forgery — «межсайтовая подделка запроса», также известна как XSRF) — вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Если жертва заходит на сайт, созданный злоумышленником, от её лица тайно отправляется запрос на другой сервер (например, на сервер платёжной

системы), осуществляющий некую вредоносную операцию (например, перевод денег на счёт злоумышленника). Для осуществления данной атаки жертва должна быть аутентифицирована на том сервере, на который отправляется запрос, и этот запрос не должен требовать какого-либо подтверждения со стороны пользователя, которое не может быть проигнорировано или подделано атакующим скриптом.

XSS (англ. Cross-Site Scripting — «межсайтовый скрипting») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «Внедрение кода».

- **Что такое DOM?**

DOM – это представление HTML-документа в виде дерева тегов.

▼ Пример

```
<!DOCTYPE HTML>
<html>
<head>
<title>О лосях</title>
</head>
<body>
Правда о лосях.
</body>
</html>
```

- **Что такое webApi ?**

- 1). Какое определение можно дать для WEB API и зачем он нужен?

Веб-API - это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения (включая любые используемые веб-фреймворки), и поэтому обычно не включает детали реализации веб-сервера или браузера, такие как SAPI или API, если они не доступны для общего доступа через удаленное веб-приложение.

Серверный веб-API - это программный интерфейс, состоящий из одной или нескольких общедоступных конечных точек для определенной системы сообщений запрос-ответ, обычно выраженной в JSON или XML, которая предоставляется через Интернет - чаще всего посредством HTTP веб сервера.

Гибридные приложения - это веб-приложения, сочетающие в себе использование нескольких серверных веб-API.

Веб-хуки - это серверные веб-API, которые принимают входные данные в виде универсального идентификатора ресурса (URI), который предназначен для использования в качестве удаленного именованного канала или типа обратного вызова, так что сервер действует как клиент для разыменования предоставленного URI и запуска событие на другом сервере, который обрабатывает это событие, тем самым обеспечивая тип однорангового IPC.

2). Можно ли сказать что если сервер на POST или GET запрос возвращает в ответ контент в формате JSON, то это у меня WEB API?

Да

3). Являются ли web-сервисами, например WCF, WEP API?

WCF и ASP.NET Web API - это фреймворк/библиотека с помощью которой вы можете организовать работу WEB-API в вашем приложении.

- **Что такое рекурсия и чем она опасна?**
- **Что будет при обработке тяжелой вычислительной функции (например цикла суммирования) и как решить возникающие проблемы?**
- **Что такое генераторы и итераторы? Что такое yield?**
- Какие есть режимы работы с файлами?

a - добавить что-то в конец файла и создать файл, если его еще нет

w - очищает файл и записывает информацию заново

r - просто чтение (если файла нет - ошибка)

- **Как устроен сборщик мусора в Python?**
Как только объекты больше не нужны, Python автоматически освобождает память из под них. Подробнее: <https://habr.com/ru/post/417215/>.
- **Возможные виды утечек памяти и как с ними бороться?**
- **Про прокси**
- **Что такое SOLID? KISS, DRY, YAGNI?**

SOLID:

- **Single Responsibility Principle** - для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.
- **Open-Closed Principle** - программные сущности ... должны быть открыты для расширения, но закрыты для модификации.
- **Liskov Substitution Principle** - объекты в программе должны быть заменяемыми на экземпляры их подтипов без изменения правильности выполнения программы.
- **Interface Segregation Principle** - много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения.
- **Dependency Inversion Principle** - зависимость на Абстракциях. Нет зависимости на что-то конкретное.

KISS (Keep It Simple Stupid) - утверждает, что большинство систем работают лучше всего, если они остаются простыми, а не усложняются.

DRY (Don't Repeat Yourself) - это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода, особенно в системах со множеством слоёв абстрагирования.

YAGNI (You Aren't Gonna Need It) - процесс и принцип проектирования ПО, при котором в качестве основной цели и/или ценности декларируется отказ от избыточной функциональности, — то есть отказ добавления функциональности, в которой нет непосредственной надобности.

- **Что такое унарный, бинарный, тернарный оператор?**
- **Что такое вычислительная сложность?**
- **Что такое область видимости?**

Обычно, по области видимости, переменные делят на глобальные и локальные. Глобальные существует в течении всего времени выполнения программы, а локальные создаются внутри методов, функций и прочих блоках кода, при этом, после выхода из такого блока переменная удаляется из памяти.

В Python их 4:

- 1). Local - эту область видимости имеют переменные, которые создаются и используются внутри функций.
- 2). Enclosing - суть данной области видимости в том, что внутри функции могут быть

вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в enclosing области видимости.

- 3). Global - Переменные области видимости global – это глобальные переменные уровня модуля (модуль – это файл с расширением .py).
- 4). Built-in - уровень Python интерпретатора. В рамках этой области видимости находятся функции open, len и т.п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. Built-in – это максимально широкая область видимости.

- **Что такое lambda функции (стрелочные функции)?**
- **Что такое мемоизация и каррирование в Python?**

▼ Мемоизация – способ оптимизации при котором сохраняется результат выполнения функции и этот результат используется при следующем вызове. (дело тут, по сути, в быстроте выполнения).

```
from functools import lru_cache #LRU - Least Recently Used
```

```
@clock  
@lru_cache()  
  
def fib(n):  
    if n < 2:  
        return n  
  
    return fib(n-2) + fib(n-1)  
  
print('fib(20) =', fib(20))
```

▼ Каррирование – это преобразование функции от многих аргументов в набор функций, каждая из которых является функцией от одного аргумента. Мы можем передать часть аргументов в функцию и получить обратно функцию, ожидающую остальные аргументы.

```
def greet_deepliy_curried(greeting):  
  
    def w_separator(separator):  
        def w_emphasis(emphasis):  
            def w_name(name):  
                print(greeting + separator + name + emphasis)  
  
            return w_name  
  
        return w_emphasis  
  
    return w_separator
```

```
    return w_name

    return w_emphasis

return w_separator

greet = greet_deepliy_curried("Hello")("...")(".")

greet('German')
greet('Ivan')
```

————через lambda————

```
greet_deepliy_curried =lambda greeting: lambda separator: lambda emphasis:
    lambda name: \
        print(greeting + separator + name + emphasis)
```

- **Что такое менеджер контекста в Python?**
- **Что такое частичное применение функции в Python?**

▼ Это процесс применения функции к части ее аргументов. Другими словами, функция, которая принимает функцию с несколькими параметрами и возвращает функцию с меньшим количеством параметров. Подробнее:
<https://habr.com/ru/post/335866/>.

```
from functools import partial

def greet(greeting, separator, emphasis, name):
    print(greeting + separator + name + emphasis)

newfunc = partial(greet, greeting='Hello', separator=',', emphasis='.')
newfunc(name='German')
newfunc(name='Ivan')
```

- **Что такое замыкание в Python?**

Замыкание (*closure*) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами. Подробнее:
<https://devpractice.ru/closures-in-python/>.

▼ Пример

```
>>> def fun1(a):
    x = a * 3
    def fun2(b):
        nonlocal x
        return b + x
    return fun2

>>> test_fun = fun1(4)

>>> test_fun(7)
19
```

- **Что такое поверхностное и глубокое копирование в Python?**

▼ Поверхностное (**shallow**) копирование - также создает отдельный новый объект или список, но вместо копирования дочерних элементов в новый объект, оно просто копирует ссылки на их адреса памяти. Следовательно, если вы сделаете изменение в исходном объекте, оно будет отражено в скопированном объекте, и наоборот.

```
# Program 2 - Shallow Copy

import copy

result_A = [95, 85, 82]

result_B = copy.copy(result_A)

print(result_A)

print(result_B)
```

▼ Глубокое (**deep**) копирование - создает новую и отдельную (независимую) копию всего объекта или списка со своим уникальным адресом памяти.

```
# Program 1 - Deep Copy

import copy

result_A = [90, 85, 82] # Student A grades

result_B = copy.deepcopy(result_A) # Student B grades (copied from A)

print(result_A)

print(result_B)
```

- **Что такое магические методы класса в Python?**

- **Что такое деструктуризация?**

Деструктуризация - распаковка, т.е. разбиение целого итерабельного объекта(например, списка) на части. Подробнее: <https://codecamp.ru/blog/python-list-destructuring-aka-packing-and-unpacking/>.

- **Что такое функциональное программирование в Python?**

- **Что такое магические методы в Python?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-magic-methods>

- **Какие есть ключевые слова в Python?**

- **Что такое регулярные выражения?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-regex>

- **Как реализуется асинхронность в Python?**

- **Как Python взаимодействует с REST API?**

- **Что такое модуль mimetypes в Python?**

- **Как происходит обработка событий в Python?**

- **Что такое worker в Python?**

- **Что такое стек вызовов в Python?**

- **Хранение данных в backend**

- **Как работают токены в Python?**

- **Что такое унарный, бинарный, тернарный операторы Python?**

- **Что такая вычислительная сложность в Python?**

- **Как работает обработка ошибок и исключений в Python?**

- **Что такое polyfill в Python?**

- **Что такое socket module?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-socket-module>

▼ Вопросы на собеседовании

1. В чем разница между модулем и пакетом в Python?

- 2. Какие встроенные типы доступны в Python?**
- 3. Что такое лямбда-функция в Python?**
- 4. Что означает пространство имен?**
- 5. Объясните разницу между списком и кортежем?**
- 6. Чем отличается pickling от unpickling?**
- 7. Что такое декораторы в Python?**
- 8. Разница между генераторами и итераторами?**
- 9. Как преобразовать число в строку?**
- 10. Как используется оператор // в Python?**
- 11. Есть ли в Python инструкция Switch или Case, как в C?**
- 12. Что такое функция range() и каковы ее параметры?**
- 13. Как используется %s?**
- 14. Обязательно ли функция Python должна возвращать значение?**
- 15. Есть ли в Python функция main()?**
- 16. Что такое GIL?**
- 17. Какой метод использовался до оператора «in» для проверки наличия ключа в словаре?**
- 18. Как изменить тип данных списка?**
- 19. Каковы ключевые особенности Python?**
- 20. Объясните управление памятью в Python**
- 21. Что такое PYTHONPATH?**
- 22. Чувствителен ли Python к регистру?**
- 23. Объясните использование функций `help()` и `dir()`.**
- 24. Что такое модули Python?**
- 25. Объясните, что означает «self» в Python.**
- 26. Что такое инженерия и процесс разработки в целом?**

27. Какие знаете принципы программирования?
28. Чем отличаются процедурная и объектов-ориентированная парадигмы программирования?
29. Какие основные принципы ООП (наследование, инкапсуляция, полиморфизм)?
30. Что такое множественное наследование?
31. Какие есть шесть этапов разработки продукта в Software Development lifecycle и какая разница между Agile и Kanban?
32. Какие есть методы HTTP-запросов и какая между ними разница?
33. Как выглядят HTTP-request / response?
34. Что такое авторизация и как она работает?
35. Что такое cookies?
36. Что такое веб уязвимость?
37. Какие знаете классические базы данных?
38. Как читать спецификацию в конкретном языке (например, PEP8 в Python)?
39. Как происходит взаимодействие клиента и сервера?
40. Какие есть подходы к проектированию API?
41. Как использовать паттерны программирования?
42. Что такое Acceptance Testing и зачем его используют?
43. Что такое модульные и интеграционные тесты, API-тесты?
44. Как писать unit-тесты?
45. Какие есть best practices в написании автотестов?
46. Какие базовые команды системы контроля версий?
47. Как использовать Git?
48. В чем разница между хешированием и шифрованием?
49. Python - интерпретируемый язык или компилируемый?
50. Какие есть меняющиеся и постоянные типы данных?

51. Что такое область видимости переменных?
52. Что такое introspection?
53. Разница между `is` и `==`?
54. Разница между `__init__()` и `__new__()`?
55. В чем разница между потоками и процессами?
56. Какие есть виды импорта?
57. Что такое класс, итератор, генератор?
58. Что такое метакласс, переменная цикла?
59. В чем разница между итераторами и генераторами?
60. В чем разница между `staticmethod` и `classmethod`?
61. Как работают декораторы, контекстные менеджеры?
62. Как работают `dict comprehension`, `list comprehension` и `set comprehension`?
63. Можно ли использовать несколько декораторов для одной функции?
64. Можно ли создать декоратор из класса?
65. Какие есть основные популярные пакеты (`requests`, `pytest`, etc)?
66. Что такое lambda-функции?
67. Что означает `*args`, `**kwargs` и как они используются?
68. Что такое `exceptions`, `<try-except>`?
69. Что такое PEP (Python Enhancement Proposal), какие из них знаете (PEP 8, PEP 484)?
70. Напишите hello-world сервис, используя один из фреймворков.
71. Какие есть типы данных и какая разница между `list` и `tuple`, зачем они?
72. Как использовать встроенные коллекции (`list`, `set`, `dictionary`)?
73. В чем заключается сложность доступа к элементам `dict`?
74. Как создается объект в Python, для чего `new`, зачем `init`?
75. Что знаете из модуля `collections`, какими еще `built-in` модулями пользовались?

76. Что такое шаблонизатор и как в нем выполнять базовые операции (объединять участки шаблона, выводить дату, выводить данные с серверной стороны)?
77. Как Python работает с HTTP-сервером?
78. Что происходит, когда создается виртуальная среда?
79. Какие есть базовые методы работы с SQL-базой данных в Python?
80. Что такое SQL-транзакция?
81. Как сделать выборку из SQL-базы с простой агрегацией?
82. Как выглядит запрос, который выполняет JOIN между таблицами и к самим себе?
83. Как отправлять запросы в SQL-базу данных без ORM?
84. Что такое алгоритмы (например, Big-O notation)?
85. Какие есть базовые алгоритмы сортировки?
86. Что такое Bubble Sort и как это работает?
87. Что такая линейная сложность сортировки?
88. Ориентируетесь ли в *nix, можете ли написать скрипты/автоматизацию для себя и коллег?
89. Что такое многопоточность?
90. Что такое архитектура веб сервисов?
91. Как работает современное нагруженное веб приложение (нарисовать и обсудить примерную архитектуру, например, Twitter или Instagram)?
92. Что нужно для сайта / сервиса среднего размера (redis \ celery \ кэш \ логирование \ метрики)?
93. Как написать, задеплоить и поддерживать (микро) сервис?
94. Как масштабировать API?
95. Як проводить Code review?
96. Что такое абстрактная фабрика, как ее реализовать и зачем ее применяют?
97. Что такая цикломатическая сложность?
98. Async Python: как работает, зачем, что под капотом?

99. Сравнить асинхронные web-фреймворки.
100. Что такое модель памяти Python?
101. Что такое SQLAlchemy (Core и ORM частей) и какие есть альтернативы?
102. Принципы работы и механизм Garbage collection, reference counting?
103. Как работает thread locals?
104. Что такое *slots*?
105. Как передаются аргументы функций в Python (by value or reference)?
106. Что такое type annotation?
107. Для чего используют нижние подчеркивания в именах классов?
108. Статические анализаторы: Flake8, Pylint, Radon.
109. Разница между SQL и NoSQL?
110. Как оптимизировать SQL-запросы?
111. Какие есть уровни изоляции транзакций?
112. Какие есть виды индексов?
113. Точечные вопросы по выбору БД, движков БД?
114. **Front-end:** есть ли опыт работы с «современным» JS (Babel, Webpack, TS, ES)?
115. **DevOps:** работали ли с Docker-контейнерами, объяснить основные термины K8s (кластер, pod, node, deployment, service), что такое Kibana?
116. **Алгоритмы:** что такое времененная сложность алгоритма (time complexity)?
117. **Углубленные знания Linux:** как зайти на внешний сервер, работать с пакетами, настроить среду и выполнять операции?
118. **Специфично для Data Science:** как работать с пакетами для обработки и визуализации данных (NumPy, Pandas и другие)?
119. Что такое @property?
120. Каким образом можно запустить код на Python параллельно?
121. Как работать с stdlib?

122. Какие задачи решали с помощью метаклассов?
123. Что такое дескрипторы?
124. Знания других языков, кроме Python (опыт).
125. Какие технологические особенности реализации распределенных систем?
126. Какие есть низкоуровневые особенности языков и фреймворков?
127. Способы и методы управления памятью.

Загуглить:

- MongoDB
- Redis
- Tarantool
- Kafka
- декомпозиция
- веб-сервисы
- сетевые протоколы
- ui-фреймворки
-



Python

Here're some Python basics you need to know

About language:

▼ What is Python used for?

- web-development (server-side)
- software development
- mathematics
- system scripting

▼ What can Python do?

- Python can be used on a server to web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

▼ Why Python?

- Python works on different platforms, OS (Windows, Mac, Linux, Raspberry, Pi, etc.).
- Python has a simple syntax to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

▼ Gear: IDE, frameworks

- IDE (Integrated Development Environment): Thonny, PyCharm, Netbeans, Eclipse.
- Frameworks: Django.

▼ Python compared to other languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation (табуляция), using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

▼ Python and Functional Programming

Python is a multi-paradigm language. It supports imperative, object oriented, as well as functional programming approach.

Functional programming paradigm recommends dividing the programming logic into set of functions that only take input and produce output not that affects the output produced for a given input.

Features of Python which help in writing functional-style programs: lambda function, recursion, iterator, generator, list comprehension, the itertools module, map(), filter(), reduce().

Python and OOP: <https://www.knowledgehut.com/tutorials/python-tutorial/python-object-oriented-programming>

Variables

Variables are containers for strong data values; they are links to data that is stored in a computer's operative memory.

A variable is created (and its type is being declared) as soon as one assigns a value to it, and can be changed anytime.

▼ Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)
y = int(3)
z = float(3)
print(x)
print(y)
print(z)
```

→

3

3

3.0

▼ Get Type

▼ You can get the type of a variable by using type() function

```
x = str(3)
y = int(3)
z = float(3)
print(type(x))
print(type(y))
print(type(z))
```

→

<class 'str'>

<class 'int'>

<class 'float'>

▼ Names

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- Multi-names can be written in: camel case (myVariableName), Pascal case (MyVariableName), snake case (my_variable_name).

▼ Assign Multiple Values

▼ Many values to multiple variables

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

→

Orange

Banana

Cherry

▼ One value to multiple variables

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

→

Orange

Orange

Orange

▼ Unpack a collection

```
fruits = ["Apple", "Pear", "Kiwi"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

→

Apple

Pear

Kiwi

▼ Output Variables

Usually - with print().

▼ Also can place several variables in one line:

```
fruits = ["Apple ", "Pear ", "Kiwi"]
x, y, z = fruits
print(x+y+z)
```

→

Apple Pear Kiwi

▼ Global Variables

▼ These are variables that are created outside of a function. They can be used everywhere, by everyone, both outside functions and inside.

```
x = "Python"

def pyfunc_my():
    print(x+" is awesome!")

pyfunc_my()
```

→

Python is awesome!

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

- ▼ To create a global variable inside a function, you can use the `global` keyword.

```
def pyfunc_my():
    global x
    x = "Python"
    print(x+" is awesome!")

pyfunc_my()

print(x+" is not that hard.")
```

→

Python is awesome!

Python is not that hard.

- ▼ Also, use the `global` keyword if you want to change a global variable inside a function.

```
x = "Java"

print(x)

def pyfunc_my():
    global x
    x = "Python"
    print(x+" is awesome!")

pyfunc_my()

print(x+" is not that hard.")
```

→

Java

Python is awesome!

Python is not that hard.

Data Types:

Integers

These are simply whole numbers. Shortened as ‘int’. E.g.: 8 and -8. (also known as “numeric type”)

When a=25 the value is called “integer literal”. Function int(“25”) - converts str to int (but it’s not integer literal).

To divide a long number, like 1000000, one can use underscores(): 1_000_000.

Floating-Point Numbers

These are numbers with a decimal point. Shortened as ‘float’. E.g.: 8.0 and -0.9 and 1.3. Max = 2×10^{400} (2e400 → inf). (also known as “numeric type”)

When a = 2.5 - floating point literal, but float(“2.5”) (converts str to float) - no.

- ▼ 1000000.0 is 1_000_000.0 is 1e6 (E[xponential] notation, here equals 1×10^6).

```
>>> 20000000000000000000.0
2e+17 #+ means that the exponent 17 is positive
```

```
>>> 1e-4
0.0001 #raised in power -4
```

Complex Numbers

Have <real part>+<imaginary part>. E.g.: 2+3j will print (2+3j) (also known as “numeric type”).

▼ Operations with numbers

“+” - addition (e.g. $2 + 3 = 5$ and $1.0 + 3 = 4.0$)

“-” - subtraction (e.g. $3 - 2 = 1$ or $4.0 - 2 = 2.0$ or just -3)

“*” - multiplication (e.g. $2 * 2 = 4$ or $2.0 * 2 = 4.0$)

“/” - division (e.g. $\text{int}(4/2) = 2$ or $4/2 = 2.0$ or $9.0/3 = 3.0$ or $\text{int}(5.0/2) = 2$)

“//” - integer division or floor division (e.g. $5//2 = 2$ or $-3//2 = -2$, b.c. it round down the number)

“**” - power exponent (e.g. $2 ** 3 = 8$ or $9 ** 0.5 = 3.0$ or $2 ** -1 = 0.5$)

▼ “%” - modulus operator (e.g. 5%3 = 2 or 6%3 = 0)

Things get a little tricky when you use the `%` operator with negative numbers:

```
>>>
```

```
>>> 5 % -3  
-1
```

```
>>> -5 % 3  
1
```

```
>>> -5 % -3  
-2
```

Although potentially shocking at first glance, these results are the product of a well-defined behavior in Python. To calculate the remainder `r` of dividing a number `x` by a number `y`, Python uses the equation `r = x - (y * (x // y))`.

For example, to find `5 % -3`, Python first finds `(5 // -3)`. Since `5 / -3` is about `-1.67`, that means `5 // -3` is `-2`. Now Python multiplies that by `-3` to get `6`. Finally, Python subtracts `6` from `5` to get `-1`.

▼ Python Assignment Operators

- “`+=`” — example: `x += 3` — same as: `x = x + 3`
- “`-=`” — example: `x -= 3` — same as: `x = x - 3`
- “`*=`” — example: `x *= 3` — same as: `x = x * 3`
- “`/=`” — example: `x /= 3` — same as: `x = x / 3`
- “`%=`” — example: `x %= 3` — same as: `x = x % 3`
- “`//=`” — example: `x //= 3` — same as: `x = x // 3`
- “`**=`” — example: `x **= 3` — same as: `x = x ** 3`
- “`&=`” — example: `x &= 3` — same as: `x = x & 3`
- “`|=`” — example: `x |= 3` — same as: `x = x | 3`
- “`^=`” — example: `x ^= 3` — same as: `x = x ^ 3`
- “`<<=`” — example: `x <<= 3` — same as: `x = x << 3`

- “`>>=`” — example: `x >>= 3` — same as: `x = x >> 3`

▼ Other Operator Groups

- Comparison: `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Logical: and, or, not.
- Identity: `is`, `is not`.
- Membership: `in`, `not in`.

▼ Bitwise

Bitwise

<u>Aa</u> Operator	≡ Name	≡ Description	📅 Date
<u>May spa</u>	may spa	Summer prep: lilac, lilly of the vally, cherry blossom vibes	@May 21, 2023
<u>March spa</u>	march spa	Women's day prep: only flowers, bubbles and bright colors	@March 5, 2023
<u>December</u>	December spa	New Year prep: vanilla, gingerbread cookies, choco and shiny winter shimmer	@December 25, 2022
<u>June spa</u>	june spa	Birthday prep: sweet cake, bright sparkles and bubbles	@June 4, 2023
<u>October</u>	October spa	Halloween prep: pumpkin'n'autumn vibes, cozy and funny	@October 30, 2022
<u>February</u>	February spa	Valentine's day prep: wine, movies, strawberries in chocolate mood	@February 12, 2023
<u>September</u>	september spa	Autumn vibes: apple pie, cinnamon roll, pancakes with maple syrup	@September 23, 2023

▼ Integer and Floating Point Methods

- ▼ `num.is_integer()` - returns True if int

```
num = 5.6
print(num.is_integer())
num1 = 5.0
print(num1.is_integer())
```

→

False

True

▼ f-string for numbers

```
>>> n = 7.125
>>> f"The value of n is {n}"
'The value of n is 7.125'
```

Those curly braces support a simple [formatting language](#) that you can use to alter the appearance of the value in the final formatted string.

```
>>> n = 7.125
>>> f"The value of n is {n:.2f}"
'The value of n is 7.12'
```

```
>>> n = 1234567890
>>> f"The value of n is {n:,}"
'The value of n is 1,234,567,890'
```

The specifier `, .2f` is useful for displaying currency values:>>>

```
>>> balance = 2000.0
>>> spent = 256.35
>>> remaining = balance - spent

>>> f"After spending ${spent:.2f}, I was left with ${remaining:.2f}"
'After spending $256.35, I was left with $1,743.65'
```

Another useful option is `%` , which is used to display percentages.

```
>>> ratio = 0.9
>>> f"Over {ratio:.1%} of Pythonistas say 'Real Python rocks!''"
"Over 90.0% of Pythonistas say 'Real Python rocks!'"

>>> # Display percentage with 2 decimal places
```

```
>>> f"Over {ratio:.2%} of Pythonistas say 'Real Python rocks!'"  
"Over 90.00% of Pythonistas say 'Real Python rocks!'"
```

Strings

These are sequences of character data. Shortened as ‘str’. E.g.: “Hey there!” or “123” or ‘’. (also known as “text type”)

If needt to include ‘ or “ in a str - “st’r”. Other - ‘st”r’.

▼ Escape Sequences ()

\' - leaves ‘ in ‘str’

\\" - leaves “ in “str”

\<new line> - newline is ignored, terminates input line

\\\ - leaves \ (backslash)

\a - ASCII Bell (**BEL**) character - nothing will be printed

\b - ASCII Backspace (**BS**) character - all in one line

\f - ASCII Formfeed (**FF**) character - new line saving all spaces

\n - new line saving all spaces

\t - tab space (horizontal)

\r - return to the begginigs of a str

\xxxx - Unicode character with 16-bit hex value **xxxx**

\Uxxxxxxxxx - Unicode character with 32-bit hex value **xxxxxxxx**

\v - ASCII Vertical Tab (**VT**) character - tab space (vertical)

\ooo - Character with octal value **ooo**

\xhh - Character with hex value **hh**

\N{name} - Character named ‘name’ in the Unicode database

Raw String - prefix ‘r’ ot ‘R’ before str - leaves all the escapes. E.g.: r’Hel\nlo’ → Hel\nlo.

Tripple-quoted str - tripple quotes enable ‘ and “ and newlines be leaft, all the escapes stay.

E.g.: “”” I’m so in love with \n\t YOU!””” →

I’m so in love with

YOU!

▼ Slicing strings

```
x = 'A fat cat sat on a mat'  
print(x[2:21:2])  
print(x[::-1])
```

→

ftctsto a
tam a no tas tac taf A

▼ String methods

▼ str.capitalize() - converts the first character to upper case

```
b = 'hey, honey!'  
print(b)  
print(b.capitalize())
```

→

hey, honey!
Hey, honey!

▼ str.casefold() - converts string into lower case

```
b = 'hey, honey!'  
print(b.upper())  
print(b.casefold())
```

→

HEY, HONEY!
hey, honey!

▼ str.center() - returns a centered string

```
b = 'hey, honey!'  
print(b.center(110, '.'))
```

→

.....hey, honey!.....

- ▼ str.count() - returns the number of times a specified value occurs in a string

```
b = 'hey, honey!'
print(b.count('h'))
print(b.count('y'))
```

→

2
2

- ▼ str.encode() - returns an encoded version of a string

```
getdefaultencoding()
# utf-8

my_string = 'кот cat'

type(my_string)
# str

my_string.encode()
# b'\xd0\xba\xd0\xbe\xd1\x82 cat'

my_string.encode('ascii')
# UnicodeDecodeError

my_string.encode('ascii', errors='ignore')
# b' cat'

my_string.encode('ascii', errors='replace')
# b'??? cat'

my_string.encode('ascii', errors='xmlcharrefreplace')
# b'ком cat'

my_string.encode('ascii', errors='backslashreplace')
```

```

# b'\u043a\u043e\u0442 cat'

my_string.encode('ascii', errors='namereplace')

# \N{CYRILLIC SMALL LETTER KA}\N{CYRILLIC SMALL LETTER O}\N{CYRILLIC SMALL LETTER TE} cat'

surrogated = '\udcd0\udcba\udcd0\udcbe\udcd1\udc82 cat'

surrogated.encode()

# UnicodeEncodeError

surrogated.encode(errors='surrogateescape')

# \xd0\xba\xd0\xbe\xd1\x82 cat'

surrogated.encode(errors='surrogatepass')

# \xed\xb3\x90\xed\xb2\xba\xed\xb3\x90\xed\xb2\xbe\xed\xb3\x91\xed\xb2\x82 cat'

```

- ▼ str.endswith() - returns True if the string ends with the specified value

```

b = 'hey, honey!'
print(b.endswith('!'))
print(b.endswith('honey!'))
print(b.endswith('world!'))

```

→

True

True

False

- ▼ str.expandtabs() - sets the tab size of the string

```

header_table = 'Name!\tRace\tWeapon\tMount'
person1 = 'Tiana\tElf\tSword\tHell Horse'
print(header_table.expandtabs(15))
print(person1.expandtabs(15))

```

→

Name!	Race	Weapon	Mount
Tiana	Elf	Sword	Hell Horse

▼ str.find() - searches the string for a specified value and returns its position

```
lyrics = "I'm completely gone from your mind\nSoon as you realize\nYou let me go  
lighter than air"  
word = lyrics.find('air')  
print(lyrics)  
print('''\nThe word "air" is number''' +str(word)+"in the poem.")
```

→

I'm completely gone from your mind
Soon as you realize
You let me go lighter than air

The word "air" is number82in the poem.

▼ str.format() - formats specified values in a string

```
def cashier(x):  
    return """Here's your check for ${}.""".format(x)  
  
print(cashier(5))
```

→

Here's your check for \$5.

▼ str.format_map() - formats specified values in a string

```
point = {'x':4, 'y':-5}  
print('{x} {y}'.format_map(point))  
  
point = {'x':4, 'y':-5, 'z': 0}  
print('{x} {y} {z}'.format_map(point))
```

→

```
4 -5  
4 -5 0
```

The `format_map()` method is similar to `str.format(mapping)` except that `str.format(**mapping)` creates a new dictionary whereas `str.format_map(mapping)` doesn't.**

- ▼ `str.index()` - searches the string for a specified value and returns its position

```
quote = "Everything is hard before it's easy."
print(quote.find('easy'))
```

→

31

- ▼ `str.isalnum()` - returns True if all characters in a string are alphanumeric

```
year = "1917"
event = "1917 revolution"
print(year.isalnum())
print(event.isalnum())
```

→

True

False

- ▼ `str.isalpha()` - returns True if all characters in a string are in the alphabet

```
army = "army"
red_army = "red army"
event = "1917 revolution"
print(army.isalpha())
print(red_army.isalpha())
print(event.isalpha())
```

→

True

False

False

- ▼ `str.isascii()` - returns True if all characters in a string are ASCII characters

```
".isascii() # True  
'wz'.isascii() # True  
'w z'.isascii() # True  
'фы'.isascii() # False  
'wzфы'.isascii() # False
```

- ▼ str.isdecimal() - returns True if all characters in a string are decimal

```
year = "1917"  
event = "1917 revolution"  
print(year.isdecimal())  
print(event.isdecimal())
```

→

```
True  
False
```

- ▼ str.isdigit() - returns True if all characters in a string are digits

```
".isdigit() # False  
' '.isdigit() # False  
'!@#'.isdigit() # False  
'abc'.isdigit() # False  
'123'.isdigit() # True  
'abc123'.isdigit() # False
```

- ▼ str.isidentifier() - returns True if the string is an identifier

```
'continue'.isidentifier() # True  
'cat'.isidentifier() # True  
'function_name'.isidentifier() # True  
'ClassName'.isidentifier() # True  
'_'.isidentifier() # True  
'number1'.isidentifier() # True
```

```
'1st'.isidentifier() # False
```

```
'*'.isidentifier() # False
```

- ▼ str.islower() - returns True if all characters in teh string are lower case

```
year = "1917"
event = "1917 revolution"
rev = "revolution"
print(year.islower())
print(event.islower())
print(rev.islower())
```

→

False

True

True

- ▼ str.isnumeric() - returns True if all characters in the string are numeric

```
".isnumeric() # False
```

```
'a'.isnumeric() # False
```

```
'0'.isnumeric() # True
```

```
'10'.isnumeric() # True
```

```
'½'.isnumeric() # True
```

```
'XII'.isnumeric() # True
```

- ▼ str.isprintable() - returns True if all characters in the string are printable

```
".isprintable() # True
```

```
' '.isprintable() # True
```

```
'1'.isprintable() # True
```

```
'a'.isprintable() # True
```

```
'█'.isprintable() # False (Group Separator)
```

```
'█'.isprintable() # False (Escape)
```

- ▼ str.isspace() - retirns True if all characters in the string are whitespaces

```
year = "19 17"
space = "      "
print(year.isspace())
print(space.isspace())
```

→

False

True

▼ str.istitle() - returns True if the string follows the rules of a title

```
title1 = "The Lonely Cat"
title2 = "The lonely cat"
title3 = "THE LONELY CAT"
title4 = "the lonely cat"
print(title1.istitle())
print(title2.istitle())
print(title3.istitle())
print(title4.istitle())
```

→

True

False

False

False

▼ str.isupper() - returns True if all the characters in the string are upper case

```
title1 = "The Lonely Cat"
title2 = "The lonely cat"
title3 = "THE LONELY CAT"
title4 = "the lonely cat"
print(title1.isupper())
print(title2.isupper())
print(title3.isupper())
print(title4.isupper())
```

→

```
False  
False  
True  
False
```

▼ `iterable.join()` - converts elements of an iterable into a string

```
list = ["like", "love", "hate"]  
x = '-'.join(list)  
print(x)  
tuple = ("like", "love", "hate")  
y = '-'.join(tuple)  
print(y)  
dict = {"me":"love", "you":"like", "she":"hate"}  
z = '-'.join(dict)  
print(z)
```

→

```
like-love-hate  
like-love-hate  
me-you-she
```

▼ `str.ljust()` - returns a left justified version of the string

```
quote = "All we need is love"  
print(quote.ljust(30, 'e'))
```

→

```
All we need is loveeeeeeeeeeee
```

▼ `str.lower()` - converts the string into lower case

```
quote = "All we need is love"  
print(quote.lower())
```

→

```
all we need is love
```

▼ `str.lstrip()` - returns a left trim version of the string

```
quote = "All we need is love"
print(quote.lstrip('All'))
print((quote.lstrip('we'))))
print((quote.lstrip('need'))))
print((quote.lstrip('is'))))
```

→

```
we need is love
All we need is love
All we need is love
All we need is love
```

- ▼ `srt.maketrans()` - returns a translation table to be used in translations

```
trans_table = str.maketrans({
    'a': 'b',
    'r': 't',
    'z':None,
})
# {97: 'b', 114: 't', 122: None}

trans_table = str.maketrans('ar', 'bt', 'z')
# {97: 98, 114: 116, 122: None}

'arroz'.translate(trans_table)
# 'btto'
```

- ▼ `str.partition()` - returns a tuple where the string is parted into three parts

```
quote = "All we need is love"
print(quote.partition(' '))
print((quote.partition('need'))))
empty = ''
print(empty.partition('.')))
numb = '.2'
print((numb.partition('.'))))
```

→

```
('All', '', 'we need is love')
('All we ', 'need', ' is love')
(' ', ' ', ' ')
(' ', '!', '2')
```

- ▼ str.replace() - returns a string where a specified value is replaced with a specified value

```
quote = "All we need is love"
print(quote.replace('love', 'money'))
```

→

All we need is money

- ▼ str.rfind() - searches the string for a specified value and returns the last position of where it was found

```
quote = "All we need is love"
print(quote.rfind('e'))
```

→

18

- ▼ str.rindex() - searches the string for a specified value and returns the last position of where it was found

```
quote = "All we need is love"
print(quote.rindex('l'))
```

→

15

- ▼ str.rjust() - returns a right justified version of the string

```
quote = "All we need is love"
a = quote.rjust(75)
print(a, '- "The Beatles" sang.')
```

→

All we need is love - "The Beatles" sang.

- ▼ str.rpartition() - returns a tuple where the string is parted into three parts

```
quote = "All we need is love"
print(quote.rpartition('love'))
```

→

('All we need is ', 'love', '')

- ▼ str.rsplit() - splits the string at the specified separator, and returns a list

```
quote = "All we need is love"
print(quote.rsplit('is'))
```

→

['All we need ', ' love']

- ▼ str.rstrip() - returns a right trim version of the string

```
quote = "All we need is love"
print(quote.rstrip('e'))
```

→

All we need is lov

- ▼ str.split() - splits the string at the specified separator, and returns a list

```
quote = "All you and I need is love and health"
print(quote.split('and'))
```

→

['All you ', ' I need is love ', ' health']

▼ str.splitlines() - splits the string at line breaks and returns a list

```
quote = "All you and I need \nis love and health"  
print(quote.splitlines())  
print(quote.splitlines(keepends=True))
```

→

```
['All you and I need ', 'is love and health']  
['All you and I need \n', 'is love and health']
```

▼ str.startswith() - returns True if a string starts with a specified value

```
quote = "All you and I need \nis love and health"  
print(quote.startswith('All'))
```

→

```
True
```

▼ str.strip() - returns a trimmed version of a string

```
print('love'.strip('l'))  
print('love'.strip('e'))  
print('love'.strip('o'))
```

→

```
ove  
lov  
love
```

▼ str.swapcase() - swaps cases

```
print('No love allowed'.swapcase())  
print('cAPS LOCK LEAP'.swapcase())
```

→

nO LOVE ALLOWED

Caps lock leap

- ▼ str.title() - converts the first character of each word into upper case

```
print('No love allowed'.title())
print('cAPS LOCK LEAP'.title())
```

→

No Love Allowed

Caps Lock Leap

- ▼ str.translate() - returns a translated string

```
trans_table = str.maketrans({'л':'к', 'д':'а', '-':None})
replaced = 'лошка -ест сметану'.translate(trans_table)
print(replaced)
```

→

кошка ест сметану

- ▼ str.upper() - converts a string into upper case

```
song = 'Love the way \nyou lie'
print(song.upper())
```

→

LOVE THE WAY

YOU LIE

- ▼ str.zfill() - fills the string with a specified number of 0 values at the beginning

```
number = '13'
specific_number = number.zfill(10)
print(specific_number)
```

→

0000000013

Boolean Type

Also called as Boolean Context and “Truthiness”, Boolean Type means that the value of an object of this type may be True or False.

They are keywords, or expressions, more precisely.

▼ E.g.:

```
def love_test(petals):
    if petals % 2 == 0:
        return True
    else:
        return False

print(love_test(5))
print(type(love_test(5)))
```

→

```
False
<class 'bool'>
```

True = 1, False = 0

▼ Counting “the” in a poem

```
>>> lines = """\
... He took his vorpal sword in hand;
...       Long time the manxome foe he sought-
... So rested he by the Tumtum tree
...       And stood awhile in thought.
...
>>> line_list = lines.splitlines()
>>> "the" in line_list[0]
False
>>> "the" in line_list[1]
True
>>> 0 + False + True # Equivalent to 0 + 0 + 1
1
>>> ["the" in line for line in line_list]
[False, True, True, False]
>>> False + True + True + False
2
>>> len(line_list)
```

```
4
>>> 2/4
0.5
```

Shorter:

```
>>> lines="""\
... He took his vorpal sword in hand;
...     Long time the manxome foe he sought-
... So rested he by the Tumtum tree
...         And stood awhile in thought.
... """ .splitlines()
>>> sum("the" in line.lower() for line in lines) / len(lines)
0.5
```

truth tables

short-circuit evaluation

The operator “**not**” (“not True” or “not False”). Another operator - “**and**” - takes two arguments (A and B), and means that the expression is False until both A and B are True. One more operator - “**or**” - is True until either A or B is True.

Comparison operators: “==” (equality); “!=” (inequality), “<” (less than, strict); “>” (greater than, strict); “<=” (less or equal, not strict), “>=” (greater or equal, not strict).

Can’t compare: str and int, dict, set. List, tuple can be compared as they are ordered lexicographically.

The operators “is” and “is not” check if A is the same object as B, or not.

The operator “in” checks an object from membership in somewhere (e.g. array).

One can chain operators: 1<3<7. Or mixed: a = 0 \ a is a<1 → True.

“None” is always False. All numbers are True, except for 0.

▼ One more value, or object in this case, evaluates to `False`, and that is if you have an object that is made from a class with a `__len__` function that returns `0` or `False`:

```
class myclass():
    def __len__(self):
```

```
return 0  
  
myobj = myclass()  
print(bool(myobj))
```

Functions

▼ Parametr or argument?

From a function's perspective:

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called.

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

▼ Arbitrary arguments

▼ If ikd how many args - *args:

```
def guests(*guests):  
    print(guests[0]+" was the first to come to my BD party!")  
  
guests("Jamie", "Anne", "Lucy", "Christoph")
```

→

Jamie was the first to come to my BD party!

▼ Keyword args

```
def family(mum, dad, me):  
    print("Hi! My name is "+me+". My mum's name is "+mum+". My dad's name is "+dad  
+".")  
family(mum = "Emily", dad="Peter", me="Clair")
```

→

Hi! My name is Clair. My mum's name is Emily. My dad's name is Peter.

▼ Arbitrary keyword args

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: `**` before the parameter name in the function definition.

▼ This way the function will receive a *dictionary* of arguments, and can access the items accordingly:

```
def detective_dairy(**info):
    print("I've got a client whose last name is"+info["lname"]+".\n")
    print("She told me to spy after her husband, "+info["hname"]+", as she thought he was cheating on her.\n")
    print("Yesterday morning I saw him leave for work, hiding "+info["present1"]+
" under suit coat.\n")
    print("In the afternoon he left his office and went to "+info["place1"]+
".\n")
    print("He came back with "+info["present2"]+".\n")
    print("Then he got a taxi to "+info["place2"]+".\n")
    print("He met "+info["person1"]+" at the entrance and greeted her as a close friend.\n")
    print("In the restaurant "+info["person2"]+" joined them.\n")
    print("After that they went to "+info["place3"]+".\n")
    print("Soon after the wife came to the same place. She realized that her husband prepared "+info["event"]+" for "+info["person3"]+".\n")
    print("That was an interesting story, even though I had to work more as "+info["proff"]+".")

print("\nHere's the first story:\n.....\n")
detective_dairy(lname="Clarson", hname="Ray", present1="a red box", place1="a flower shop", present2="a huge bouquet of red roses", place2="restaurant", person1="a red-haired girl", person2="a brightly dressed brunette", place3="his house", event="a birthday party", person3="her", proff="a photographer")

print("\nHere's the second story:\n.....\n")
detective_dairy(lname="Lanson", hname="Sam", present1="a black box", place1="a store", present2="a medium package", place2="a park", person1="a man in a blue suit", person2="a man in a white suit", place3="his house", event="a trap", person3="her", proff="a police officer")

print("\n.....")
print("\nIndeed, how a few details can change the whole picture! Fascinating, isn't it?")
```

→

Here's the first story:.....

I've got a client whose last name isClarson.

She told me to spy after her husband, Ray, as she thaught he was cheating on her.

Yesterday morning I saw him leave for work, hiding a red box under suit coat.

In the afternoon he left his office and went to a flower shop.

He came back with a huge bouquet of red roses.

Then he got a taxi to restaurant.

He met a red-haired girlat the entrance and greeted her as a close friend.

In the restaurant a brightly dressed brunettejoined them.

After that they went to his house.

Soon after the wife came to the same place. She realized that her husband prepared a birthday partyforher.

That was an interesting story, even though I had to work more as a photographer.

Here's the second story:.....

I've got a client whose last name isLanson.

She told me to spy after her husband, Sam, as she thaught he was cheating on her.

Yesterday morning I saw him leave for work, hiding a black box under suit coat.

In the afternoon he left his office and went to a store.

He came back with a medium package.

Then he got a taxi to a park.

He met a man in a blue suitat the entrance and greeted her as a close friend.

In the restaurant a man in a white suitjoined them.

After that they went to his house.

Soon after the wife came to the same place. She realized that her husband prepared a trapforher.

That was an interesting story, even though I had to work more as a police officer.

.....

Indeed, how a few details can change the whole picture! Fascinating, isn't it?

▼ Default parametr value

```
def attendant_country(country=" Russia"):
    print("- Hi! Where are you from?")
    print("- Hi! I'm from"+country+". Great conference, isn't it?")
    if country == "Russia":
        print("- Yes! And which city are you from?")
    else:
        print("- Indeed! How do you like our city? Have you been sightseeing?")

print("-----Dialogue 1-----")
attendant_country()
print()
print("-----Dialogue 2-----")
attendant_country(country=" the USA")
```

→

-----Dialogue 1-----

- Hi! Where are you from?
- Hi! I'm from Russia. Great conference, isn't it?
- Yes! And which city are you from?

-----Dialogue 2-----

- Hi! Where are you from?
- Hi! I'm from the USA. Great conference, isn't it?
- Indeed! How do you like our city? Have you been sightseeing?

▼ Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

▼ E.g. if you send a List as an argument, it will still be a List when it reaches the function

```
def words(wordlist):
    for x in wordlist:
        print(x)

vocab = ["a cat", "a mouse", "a house"]
words(vocab)
```

→

a cat
a mouse
a house

▼ return

▼ To let a function return a value, use the `return` statement:

```
def myfunc(x):
    return x**5

print(myfunc(1))
print(myfunc(2.1))
print(myfunc(-3))
```

→

1
40.84101000000001
-243

▼ pass

`function` definitions cannot be empty, but if you for some reason have a `function` definition with no content, put in the `pass` statement to avoid getting an error.

▼ Recursion

Recursion means that a function can call itself, which enables us to loop through data to reach a result.

▼ Example

In this example, `tri_recursion()` is a function that we have defined to call itself ("recurse"). We use the `k` variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

```
def tri_recursion(k):
    if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
    else:
        result = 0
```

```
    return result

print("Recursion Example Results")
tri_recursion(6)
```

→

Recursion Example Results

```
1
3
6
10
15
21
```



▼ Explanation

Try tracing the function with a pencil and paper. In this case, the print statement inside the function may be a bit misleading.

Consider this part of the program,

```
if(k>0):
    result = k+tri_recursion(k-1)
    ...
```

From here,

```
tri_recursion(6) = 6 + tri_recursion(5)
```

So to get the result for `tri_recursion(6)` we must get the result of `tri_recursion(5)`. Following this logic, the problem reduces to:

```
tri_recursion(6)
= 6 + tri_recursion(5)
= 6 + 5 + tri_recursion(4)
= 6 + 5 + 4 + tri_recursion(3)
= 6 + 5 + 4 + 3 + tri_recursion(2)
= 6 + 5 + 4 + 3 + 2 + tri_recursion(1)
= 6 + 5 + 4 + 3 + 2 + 1 + tri_recursion(0)
```

Now notice that 0 is not greater than 0 so the program moves to the body of the else clause:

```
else:
    result = 0
...
```

Which means `tri_recursion(0) = 0`. Therefore:

```
tri_recursion(6)
= 6 + 5 + 4 + 3 + 2 + 1 + tri_recursion(0)
= 6 + 5 + 4 + 3 + 2 + 1 + 0
= 21
```

Points to note:

1. In running this program, `k` is never equal to `1`, infact it is impossible.
2. It is misleading to think of control flow in terms of "the compiler moving across a program". The compiler doesn't do anything during execution (JIT is a different matter). It is better to think in terms of control flow / order of execution in procedural languages, equationally in functional programming and relations in logic programming.

▼ Lambda

It's an anonymous function, that can take any number of arguments, but can only have one expression.

▼ Syntax: `lambda arguments : expression`

```
x = lambda a, b : a/b
print(x(2, 3))
print(x(5, -10))
```

→

0.6666666666666666

-0.5

▼ Lambda can be used inside another function

```
def myfunc(n):
    return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

→

22

33

In Python there are some **Built-in functions:**

▼ Math

▼ `abs()` - returns absolute value of a number

```
print(abs(-26))
print(abs(0.2))
print(abs(83))
print(abs(-3.09))
```

→

26

0.2

83

3.09

- ▼ `divmod()` - returns quotient and remainder of integer division

```
print(divmod(45, 5))
print(divmod(33, 2))
```

→

(9, 0)

(16, 1)

- ▼ `max()` - returns the largest of given arguments or items in an iterable

```
list = [-3, 0, 1, 2, 4, 5, 7, 8]
print(max(list))
```

→

8

- ▼ `min()` - returns the smallest of given arguments or items in an iterable

```
list = ['lol', 'love']
list1 = ['a', 'b']
list2 = ['1', '2', 'a', 'b']
print(min(list))
print(min(list1))
print(min(list2))
```

→

lol

a

1

```
list3 = [1, 2, 'a', 'b']
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print(min(list3))

TypeError: '<' not supported between instances of 'str' and 'int'

```
tuple = (1, 2, 'love', 'a')
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3, in <module>
print(min(tuple))

TypeError: '<' not supported between instances of 'str' and 'int'

```
dict = {1:"May", "cat":13, 2:19, "mouse":"grey"}
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print(min(dict))

TypeError: '<' not supported between instances of 'str' and 'int'

```
tuple = ('love', 'a')
dict = {"mouse":"grey"}
print(min(tuple))
print(min(dict))
```

→

a

mouse

▼ pow() - raises a number to a power

```
print(pow(4, 2))
print(pow(-0.3, 3))
```

→

```
16  
-0.02699999999999996
```

▼ round() - rounds floating-point value

```
print(round(5.3))  
print(round(5.5))  
print(round(5.6))  
print(round(-5.3))  
print(round(-5.5))  
print(round(-5.6))
```

→

```
5  
6  
6  
-5  
-6  
-6
```

▼ sum() - sums the items of an iterable

```
list = [1, 2, -3, 5]  
tuple = (9, 8, 0.5)  
print(sum(list))  
print(sum(tuple))
```

→

```
5  
17.5
```

```
dict = {1:2, 7:4}  
print(sum(dict))
```

→ sums only keys

8

▼ Type Conversion

- ▼ `ascii()` - returns a string containing a printable representation of an object
- ▼ `bin()` - converts an integer to a binary string

```
print(bin(0))
print(bin(1))
print(bin(2))
print(bin(3))
print(bin(4))
print(bin(5))
print(bin(6))
print(bin(7))
print(bin(8))
print(bin(9))
print(bin(10))
print(bin(11))
print(bin(-1))
```

→

```
0b0
0b1
0b10
0b11
0b100
0b101
0b110
0b111
0b1000
0b1001
0b1010
0b1011
-0b1
```

- ▼ `bool()` - converts an argument to a Boolean value

```
print(bool(1))
print(bool(0))
print(bool(5))
print(bool(-0.4))
print(bool('lol'))
```

```
list = [0]
print(bool(list))
list1 = []
print(bool(list1))
```

→

True
False
True
True
True
True
False

- ▼ chr() - returns representation of character given by integer argument

```
print(chr(65))
print(chr(93))
```

→

A
J

Given an integer from 0 to 255, chr() returns str with the relevant character from ASCII table.

- ▼ complex() - returns a complex number constructed from arguments

complex() returns a number with two parts - real and imaginary (with j)

```
print(complex(2, -0.3))
print(complex(4))
```

→

(2-0.3j)
(4+0j)

- ▼ float() - returns a floating-point object constructed from a number or string

```
print(float(9))
print(float(0))
print(float("12.3"))
```

→

9.0

0.0

12.3

▼ `hex()` - converts an integer to a hexadecimal string

```
print(hex(5))
print(hex(277))
print(hex(-48))
print(hex(0))
```

→

0x5

0x115

-0x30

0x0

▼ `int()` - returns an integer object constructed from a number or a string

```
print(int('23'))
print(int(2.3))
print(int(-2.3))
```

→

23

2

-2

▼ `oct()` - converts an integer to an octal string

```
print(oct(8))
print(oct(-8))
```

→

0o10
-0o10

▼ **ord()** - returns integer representation of a character

The opposite to **chr()**

```
print(ord('A'))  
print(ord('a'))  
print(ord('z'))  
print(ord('Z'))
```

→

65
97
122
90

▼ **repr()** - returns a string containing a printable representation of an object

```
list = (0.1, -2)  
print(repr(1))  
print(repr('Lol'))  
print(repr(list))
```

→

1
'Lol'
(0.1, -2)

▼ **str()** - returns a string version of an object

```
list = (0.1, -2)  
dict = {"panda":23, 1:"bear"}  
print(str(1))  
print(repr(dict))  
print(str(list))  
print(type(str(dict)))
```

```
→  
1  
{'panda': 23, 1: 'bear'}  
(0.1, -2)  
<class 'str'>
```

- ▼ `type()` - returns the type of an object or creates a new type object

```
dict = {"panda":23, 1:"bear"}  
print(type(dict))
```

```
→  
<class 'dict'>
```

`type()` can also be used to create a class faster. These two methods create the same object:

```
>>>class Foo(object):  
...     bar =True
```

and

```
Foo = type('Foo', (), {'bar':True})
```

- ▼ Iterables and Iterators

- ▼ `all()` - returns True if all elements of an iterable are true

```
list = [0, 1, 2, "lol"]  
list1 = [1, 2, "lol"]  
print(all(list))  
print(all(list1))
```

```
→  
False  
True
```

- ▼ `any()` - returns True if any elements of an iterable are true

```
list = [0]
list1 = [0, 1, 2, "lol"]
print(any(list))
print(any(list1))
```

→

False

True

- ▼ `enumerate()` - returns a list of tuples containing indices and values from an iterable

```
list = [-1, 0, 1, 2, 3, 4]
for i in enumerate(list):
    print(i)
```

→

(0, -1)
(1, 0)
(2, 1)
(3, 2)
(4, 3)
(5, 4)

The first element of a tuple is index of an element of an iterable, and the second - its value.

- ▼ `filter()` - filters elements from an iterable

E.g.:

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
print(list(filter(lambda x: x[0].lower() in 'aieou', creature_names)))
```

→

['Odrey']

Read as:

I need to print a list

that is a result of function filter()

that is performed with usage of function lambda(),

where every element of an iterable is represented as x;

x[0] enables lambda() to go through every first letter of every word in the list creature_names,

lower - makes the first letters written in lowers case so that they were the same as our example ‘aieou’;

in the end I mention th elist where I need there operations to be done.

▼ iter() - returns an iterator object

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
iterated_creature_names = iter(creature_names)
print(next(iterated_creature_names))
print(next(iterated_creature_names))
print(next(iterated_creature_names))
print(next(iterated_creature_names))
```

→

Lola

Lilu

Jim

Odrey

The number of iterator object call outs must not exceed the number of objects in the iterable.

▼ len() - returns the length of an object

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
name = "Kiki"
print(len(creature_names))
print(len(name))
```

→

4

4

Doesn't work with int and float.

- ▼ map() - applies a function to every item of an iterable

```
list1 = [1, 2, 3, 4]
mapped_list = list(map(lambda x: x**5-1, list1))
print(mapped_list)
```

→

[4, 9, 14, 19]

```
aquarium_creatures = [
    {"name": "sammy", "species": "shark", "tank number": 11, "type": "fish"},
    {"name": "ashley", "species": "crab", "tank number": 25, "type": "shellfish"},
    {"name": "jo", "species": "guppy", "tank number": 18, "type": "fish"},
    {"name": "jackie", "species": "lobster", "tank number": 21, "type": "shellfish"},
    {"name": "charlie", "species": "clownfish", "tank number": 12, "type": "fish"},
    {"name": "olly", "species": "green turtle", "tank number": 34, "type": "turtle"}
]
def assign_to_tank(aquarium_creatures, new_tank_number):
    def apply(x):
        x["tank number"] = new_tank_number
        return x
    return map(apply, aquarium_creatures)
print(list(assign_to_tank(aquarium_creatures, 42)))
```

→

```
[{"name": "sammy", "species": "shark", "tank number": 42, "type": "fish"}, {"name": "ashley", "species": "crab", "tank number": 42, "type": "shellfish"}, {"name": "jo", "species": "guppy", "tank number": 42, "type": "fish"}, {"name": "jackie", "species": "lobster", "tank number": 42, "type": "shellfish"}, {"name": "charlie", "species": "clownfish", "tank number": 42, "type": "fish"}, {"name": "olly", "species": "green turtle", "tank number": 42, "type": "turtle"}]
```

```
base = [1, 2, 3]
powers = [1, 2, 3]
powered_numbers = list(map(pow, base, powers))
print(powered_numbers)
```

→

[1, 4, 27]

- ▼ next() - retrieves the next item from an iterator

```
numb = iter([1, 3, 5])
print(next(numb))
print(next(numb))
print(next(numb))
```

→

1
3
5

- ▼ range() - generates a range of integer values

```
for item in range(10):
    print(item)
```

→

0
1
2
3
4
5
6
7
8
9

▼ reversed() - returns a reverse iterator

```
numb = [1, 2, 3]
print(list(reversed(numb)))
```

→

[3, 2, 1]

▼ slice() - returns a slice object

```
quote = "Per aspera ad astra"
a = slice(4, 10)
b = slice(0, 3)
print(quote[a])
print(quote[b])
```

→

aspera

Per

▼ sorted() - returns a sorted list from an iterable

```
signs = ["Aries", "Taurus", "Gemini", "Cancer", "Leo", "Virgo", "Libra", \
         "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pieces"]
print(sorted(signs))
```

→

['Aquarius', 'Aries', 'Cancer', 'Capricorn', 'Gemini', 'Leo', 'Libra', 'Pieces', 'Sagittarius',
'Scorpio', 'Taurus', 'Virgo']

▼ zip() - creates an iterator that aggregates elements from iterables

```
signs = ["Aries", "Taurus", "Gemini", "Cancer", "Leo", "Virgo", "Libra", \
         "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pieces"]
dates = ["(21st March - 20th April)", "(21st April - 21st May)", "(22nd May - 21st June)", "(22nd June - 22nd July)", "(23rd July - 23rd August)", "(24th August - 22nd September)", "(23rd September - 23rd October)", "(24th October - 22nd November)", "(23rd November - 21st December)", "(22nd December - 20th January)", "(21st January - 18th February)", "(19th February - 20th March)"]
```

```
for s, d in zip(signs, dates):
    print(s, d)
```

→

Aries (21st March – 20th April)
Taurus (21st April - 21st May)
Gemini (22nd May - 21st June)
Cancer (22nd June - 22nd July)
Leo (23rd July - 23rd August)
Virgo (24th August - 22nd September)
Libra (23rd September - 23rd October)
Scorpio (24th October - 22nd November)
Sagittarius (23rd November - 21st December)
Capricorn (22nd December - 20th January)
Aquarius (21st January - 18th February)
Pisces (19th February - 20th March)

▼ If/elif/else

▼ if...else...

```
French_vocab = {
    "food" : {
        "le fromage":"cheese",
        "le lait":"milk"
    },
    "animals" : {
        "le chat":"a cat",
        "l'elephant":"an elephant"
    },
    "time" : {
        "le soir": "tonight",
        "le matin":"this morning"
    }
}

def Antoshka():
    x = input('Введите слово: ')
    if x in French_vocab["animals"] or x in French_vocab["food"] or x in French_vocab["time"]:
        print("О, я знаю это слово! Но я забыл...")
    else:
        print("Это мы не проходили, это нам не задавали...")
```

```
Antoshka()
```

→

[input] le chat

[output] О, я знаю это слово! Но я забыл...

▼ elif

Stands in between:

```
French_vocab = {  
    "food" : {  
        "le fromage": "cheese",  
        "le lait": "milk"  
    },  
    "animals" : {  
        "le chat": "a cat",  
        "l'elephant": "an elephant"  
    },  
    "time" : {  
        "le soir": "tonight",  
        "le matin": "this morning"  
    }  
}  
  
def Antoshka():  
    x = input('Введите слово: ')  
    if x in French_vocab["food"] :  
        print("О, это что-то съедобное!")  
    elif x in French_vocab["animals"] or x in French_vocab["time"]:  
        print("Не знаю, что это, но точно не еда!")  
    else:  
        print("Это мы не проходили, это нам не задавали...")  
  
Antoshka()
```

→

[input] le chat

[output] Не знаю, что это, но точно не еда!

▼ Shorthand if

```
bag_items = ["sun_screen", "bottle_of_water", "magazine", "phone"]
x = input("Есть ли у Насти:")
if x in bag_items: print("Да, у Насти это есть!")
```

▼ Shorthand if...else...

This technique is known as **Ternary Operators**, or **Conditional Expressions**

```
bag_items = ["sun_screen", "bottle_of_water", "magazine", "phone"]
x = input("Есть ли у Насти:")
print("Да, у Насти это есть!") if x in bag_items else print("Такого нет!")
```

▼ Multiple conditions

```
x = int(input("Введите целое число: "))
print("Число больше нуля.") if x>0 else print("Число равно нулю.") if x==0 else print("Число меньше нуля.")
```

▼ Nested if

```
print('Добро пожаловать в нашу игру "Великолепный век!"')
x = input("Выберите пол персонажа: ").title()
if x == "Мужчина":
    y = int(input("Сколько вы задонатили в игру? Введите целое число: "))
    if y >= 1000:
        print("Поздравляем! Вы - султан! Вся власть в ваших руках!")
    else:
        print("Вы - слуга. Стоит задонатить еще, чтобы стать султаном!")
else:
    z = input("Напишите ваше главное качество: ").title()
    while z != "Красивая":
        s = input("Выберите другое качество: ").title()
        z //="Красивая"
    print("Подравляем! Вы в гареме!")
```

▼ pass statement

`if` statements cannot be empty, but if you for some reason have an `if` statement with no content, put in the `pass` statement to avoid getting an error.

```
a = 33  
b = 200  
if b > a:  
    pass
```

▼ While loop

▼ With the while loop we can execute a set of statements as long as a condition is true.

Note: remember to increment i, or else the loop will continue forever.

```
a = 0  
while a!=10:  
    print(a)  
    a = a+1
```

→

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

▼ break - stops the loop at the condition

```
a = 0  
while a!=10:  
    print(a)  
    if a == 5:  
        break  
    a = a+1
```

→

```
0  
1  
2  
3  
4  
5
```

- ▼ continue - stops prev. iteration and continues with the next one

```
i = 1  
while i < 6:  
    i *= 2  
    if i == 3:  
        continue  
    print(i)
```

→

```
2  
4  
8
```

- ▼ else

```
i = 1  
while i < 6:  
    print(i)  
    i *= 2  
else:  
    print("The end")
```

→

```
1  
2  
4  
The end
```

x//= “something”- another way to break the loop

▼ for loop

▼ loop

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    print(i)
```

→

a floor
windows
a door
a ceiling
walls

▼ break

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    print(i)
    if i == "a door":
        break
```

→

a floor
windows
a door

▼ continue

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    if i == "windows":
        continue
    print(i)
```

→

a floor
a door

a ceiling
walls

▼ range()

```
for x in range(10):
    print(x*3)
    x = x+1
```

→

0
3
6
9
12
15
18
21
24
27

▼ else

```
for x in range(5, 10, 2):
    print(x*3)
    x = x+1
else:
    print("Finish")
```

→

15
21
27
Finish

```
for x in range(5, 10):
    if x == 8: break
    print(x)
```

```
    else:  
        print("Finish")
```

→

5
6
7

▼ nested loops

▼ The "inner loop" will be executed one time for each iteration of the "outer loop":

```
girls = ["Sara", "Rachel", "Emily", "Lola"]  
boys = ["Sam", "Michael", "Jack", "Brandon"]  
  
def couples():  
    for x in girls:  
        for y in boys:  
            print(str(x), "+", str(y), "= a nice couple!")  
  
couples()
```

→

Sara + Sam = a nice couple!
Sara + Michael = a nice couple!
Sara + Jack = a nice couple!
Sara + Brandon = a nice couple!
Rachel + Sam = a nice couple!
Rachel + Michael = a nice couple!
Rachel + Jack = a nice couple!
Rachel + Brandon = a nice couple!
Emily + Sam = a nice couple!
Emily + Michael = a nice couple!
Emily + Jack = a nice couple!
Emily + Brandon = a nice couple!
Lola + Sam = a nice couple!
Lola + Michael = a nice couple!

Lola + Jack = a nice couple!
Lola + Brandon = a nice couple!

▼ pass

`for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.

```
for x in [0, 1, 2]:  
    pass
```

Composite Data Type

▼ `bytearray()` - creates and returns an object of the `bytearray` class (also known as “binary type”)

```
>>>b = bytearray(b'hello world!')  
>>>b  
bytearray(b'hello world!')  
>>>b[0]  
104  
>>>b[0] = b'h'  
Traceback (most recent call last):  
  File "", line 1, in  
    b[0] = b'h'  
TypeError: an integer is required  
>>>b[0] = 105  
>>>b  
bytearray(b'iello world!')  
>>>for i in range(len(b)):  
...     b[i] += i  
...>>>b  
bytearray(b'ifnos%}vzun,')
```

▼ `bytes()` - creates and returns a `bytes` object (similar to `bytearray` but immutable) (also known as “binary type”)

```
>>>b 'bytes'  
b'bytes'  
>>>'Байты'.encode('utf-8')  
b'\xd0\x91\xd0\xbd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'  
>>>bytes('bytes', encoding = 'utf-8')  
b'bytes'
```

```
>>>bytes([50, 100, 76, 72, 41])  
b'2dLH')
```

- ▼ dict() - creates a dict object (also known as “mapping type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",  
Taurus = "21st April - 21st May",  
Gemini = "22nd May - 21st June",  
Cancer = "22nd June - 22nd July",  
Leo = "23rd July - 23rd August",  
Virgo = "24th August - 22nd September",  
Libra = "23rd September - 23rd October",  
Scorpio = "24th October - 22nd November",  
Sagittarius = "23rd November - 21st December",  
Capricorn = "22nd December - 20th January",  
Aquarius = "21st January - 18th February",  
Pieces = "19th February - 20th March")  
print(zodiac_signs)  
print(zodiac_signs['Leo'])
```

→

```
{'Aries': '21st March – 20th April', 'Taurus': '21st April - 21st May', 'Gemini': '22nd May - 21st June', 'Cancer': '22nd June - 22nd July', 'Leo': '23rd July - 23rd August', 'Virgo': '24th August - 22nd September', 'Libra': '23rd September - 23rd October', 'Scorpio': '24th October - 22nd November', 'Sagittarius': '23rd November - 21st December', 'Capricorn': '22nd December - 20th January', 'Aquarius': '21st January - 18th February', 'Pieces': '19th February - 20th March'}  
23rd July - 23rd August
```

- ▼ frozenset() - creates a frozenset object (it's like set, but immutable)

```
b = frozenset('qwerty')  
print(b.add(1))
```

→

```
Traceback (most recent call last):  
File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 2, in <module>  
print(b.add(1))  
AttributeError: 'frozenset' object has no attribute 'add'
```

▼ `list()` - creates a list object (also known as “sequence type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(list(zodiac_signs))
```

→

```
['Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo', 'Libra', 'Scorpio', 'Sagittarius', 'Capricorn',
'Aquarius', 'Pieces']
```

▼ `object()` - creates a new featureless object

You cannot add new properties or methods to this object. This object is the base for all classes, it holds the built-in properties and methods which are default for all classes.

```
test = object()

print(type(test))
print(dir(test))
```

→

```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__',
 '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__']
```

In the program, we have used `type()` to get the type of the object. Similarly, we have used `dir()` to get all the attributes. These attributes (properties and methods) are common to instances of all Python classes.

▼ `set()` - creates a set object (also known as “set type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(set(zodiac_signs))
```

→

```
{'Scorpio', 'Gemini', 'Pieces', 'Virgo', 'Aquarius', 'Leo', 'Sagittarius', 'Aries', 'Libra', 'Taurus',
'Cancer', 'Capricorn'}
```

▼ `tuple()` - creates a tuple object (also known as “sequence type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(tuple(zodiac_signs))
```

→

```
('Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo', 'Libra', 'Scorpio', 'Sagittarius', 'Capricorn',
'Aquarius', 'Pieces')
```

▼ `*memoryview` (also known as “binary type”) -

▼ `** range` (also known as “sequence type”) -

Arrays

▼ List (lis') - an ordered mutable collection of objects of various types (str and int cannot be in one list in Python3).

▼ Let's create a list:

- Way 1:

```
a = list('Morning')
print(a)
```

→

['M', 'o', 'r', 'n', 'i', 'n', 'g']

- Way 2:

```
a = [1, 2, 3]
print(a)
```

→

[1, 2, 3]

- Way 3:

```
list_generator = [c*3 for c in range(3)]
print(list_generator)
```

→

[0, 3, 6]

▼ List Methods

▼ list.append(x) - adds an elements to the end of a list

```
sample = [1, 2, 3]
sample.append(4)
```

```
print(sample)
sample.append(5)
print(sample)
```

→

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
```

- ▼ `list.extend(L)` - extends a list, adding elements from the list L to the end

```
sample = [1, 2, 3]
another_sample = [4, 5, 6]
sample.extend(another_sample)
print(sample)
print(another_sample)
```

→

```
[1, 2, 3, 4, 5, 6]
[4, 5, 6]
```

- ▼ `list.insert(i, x)` - pastes an x-element to the i-position in the list

```
sample = [1, 2, 3]
print(sample)
sample.insert(3, 4)
print(sample)
```

→

```
[1, 2, 3]
[1, 2, 3, 4]
```

- ▼ `list.remove(x)` - deletes the first x-element (if none - ValueError)

```
sample = [1, 2, 3, 2]
print(sample)
sample.remove(2)
print(sample)
```

→

[1, 2, 3, 2]

[1, 3, 2]

- ▼ list.pop(i) - deletes the element with i-index and returns the list. If list.pop() - deletes the last element

```
sample = [1, 2, 3, 2]
print(sample)
sample.pop(2)
print(sample)
```

→

[1, 2, 3, 2]

[1, 2, 2]

- ▼ list.index(x, [start [, end]]) - returns the index of the first x-element (searches from start to end)

```
sample = [1, 2, 2, 2, 3, 2]
print(sample)
print(sample.index(2))
```

→

[1, 2, 2, 2, 3, 2]

1

- ▼ list.count(x) - returns quantity of x-elements

```
sample = [1, 2, 2, 2, 3, 2]
print(sample)
print(sample.count(2))
```

→

[1, 2, 2, 2, 3, 2]

4

- ▼ list.sort(key=function/ None, reverse = True/ False) - sorts out the list according to the function given

```
sample_list = ['Rose', 'Lily', 'Sharon', 'Mio']
sample_list.sort()
print(sample_list)
sample_list.sort(reverse=True)
print(sample_list)

def sample_func (x):
    return len(x)
sample_list.sort(key=sample_func, reverse=True)
print(sample_list)
```

→

```
['Lily', 'Mio', 'Rose', 'Sharon']
['Sharon', 'Rose', 'Mio', 'Lily']
['Sharon', 'Rose', 'Lily', 'Mio']
```

▼ `list.reverse()` - returns a reversed list

```
sample_list = ['а р о з а у п а л а н а р у к у а з о р а']
sample_list.reverse()
print(sample_list)
sample_list.reverse()
print(sample_list)
```

→

```
['а р о з а у п а л а н а р у к у а з о р а']
['а р о з а у п а л а н а р у к у а з о р а']
```

▼ `list.copy()` - makes a shallow copy of a list

```
# mixed list
prime_numbers = [2, 3, 5]

# copying a list
numbers = prime_numbers.copy()

print('Copied List:', numbers)

# Output: Copied List: [2, 3, 5]
```

Another way to make a copy is to use the built-in method `list()`

```
thislist = ["apple", "banana", "cherry"]
```

```
mylist = list(thislist)
```

```
print(mylist)
```

▼ list.clear() - clears up a list

```
sample_list = ['а р о з а у п а л а н а р у к у а з о р а']  
sample_list.reverse()  
print(sample_list)  
sample_list.clear()  
print(sample_list)
```

→

```
['а р о з а у п а л а н а р у к у а з о р а']  
[]
```

List methods change the list itself, so we don't need to use a variable for the changed list.

▼ Looping a list

▼ Loop through a list

```
thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
    print(x)
```

→

```
apple  
banana  
cherry
```

▼ Loop by index

```
thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
    print(thislist[i])
```

→

```
apple  
banana  
cherry
```

▼ While Loop

```
thislist = [1, 2, 3, 4, 5, 6 ,7 ,8, 9]  
i = 0  
while i < len(thislist):  
    print(thislist[i])  
    i = i + 1
```

```
→  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

▼ Looping Using List Comprehension

```
thislist = ["apple", "banana", "cherry"]  
[print(x) for x in thislist]
```

```
→  
apple  
banana  
cherry
```

▼ List Comprehencion

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Syntax: newlist = [*expression* for *item* in *iterable* if *condition* == True]

The *condition* is like a filter that only accepts the items that evaluate to `True`.

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list.

▼ Without:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

→

`['apple', 'banana', 'mango']`

▼ With:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

→

`['apple', 'banana', 'mango']`

▼ More examples:

```
#Accept only numbers lower than 5:
newlist = [x for x in range(10) if x < 5]
print(newlist)
```

→

`[0, 1, 2, 3, 4]`

```

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x.upper() for x in fruits] #Set the values in the new list to upper case:

print(newlist)

newlist = ['hello' for x in fruits] #Set all values in the new list to 'hello':

print(newlist)

newlist = [x if x != "banana" else "orange" for x in fruits] #Return "orange" instead of "banana":

print(newlist)

```

→

```

['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
['hello', 'hello', 'hello', 'hello', 'hello']
['apple', 'orange', 'cherry', 'kiwi', 'mango']

```

▼ Nested

▼ Nested loops can also be used in a list comprehension expression. To obtain list of all combinations of items from two lists:

```

print([{x:y} for x in range(1, 15, 2) for y in range(1, 25, 2)])
def cons(word):
    result = [char for char in word if char not in ["a", "e", "i", "o", "u"]]
    print(result)
cons(word="siblings")
cons(word="country")

```

→

```

[{:1: 1}, {:1: 3}, {:1: 5}, {:1: 7}, {:1: 9}, {:1: 11}, {:1: 13}, {:1: 15}, {:1: 17}, {:1: 19}, {:1: 21}, {:1: 23}, {:3: 1}, {:3: 3}, {:3: 5}, {:3: 7}, {:3: 9}, {:3: 11}, {:3: 13}, {:3: 15}, {:3: 17}, {:3: 19}, {:3: 21}, {:3: 23}, {:5: 1}, {:5: 3}, {:5: 5}, {:5: 7}, {:5: 9}, {:5: 11}, {:5: 13}, {:5: 15}, {:5: 17}, {:5: 19}, {:5: 21}, {:5: 23}, {:7: 1}, {:7: 3}, {:7: 5}, {:7: 7}, {:7: 9}, {:7: 11}, {:7: 13}, {:7: 15}, {:7: 17}, {:7: 19}, {:7: 21}, {:7: 23}, {:9: 1}, {:9: 3}, {:9: 5}, {:9: 7}, {:9: 9}, {:9: 11}, {:9: 13}, {:9: 15}, {:9: 17},

```

```
{9: 19}, {9: 21}, {9: 23}, {11: 1}, {11: 3}, {11: 5}, {11: 7}, {11: 9}, {11: 11},  
{11: 13}, {11: 15}, {11: 17}, {11: 19}, {11: 21}, {11: 23}, {13: 1}, {13: 3},  
{13: 5}, {13: 7}, {13: 9}, {13: 11}, {13: 13}, {13: 15}, {13: 17}, {13: 19},  
{13: 21}, {13: 23}]  
['s', 'b', 'T', 'n', 'g', 's']  
['c', 'n', 't', 'r', 'y']
```

▼ Join Lists

▼ Concatenation

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
animals = ["koala", "bear", "flying fox", "rabbit", "turtle"]  
  
animals_and_their_food = fruits + animals  
print(animals_and_their_food)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'koala', 'bear', 'flying fox', 'rabbit', 'turtle']
```

▼ .append() + for

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
more_fruits = ["pear", "dragon fruit", "pineapple"]  
  
for x in more_fruits:  
    fruits.append(x)  
  
print(fruits)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'pear', 'dragon fruit', 'pineapple']
```

▼ .extend()

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
more_fruits = ["pear", "dragon fruit", "pineapple"]
```

```
fruits.extend(more_fruits)  
print(fruits)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'pear', 'dragon fruit', 'pineapple']
```

▼ Tuple (tuple) - an ordered immutable collection of objects (like list but immutable).

Immutability makes a tuple more secure than a list: it can't be changed neither intentionally (which is also sad) nor unintentionally (which is great).

Moreover, a tuple weights less, it is processed faster , and it can be used as a key in a dictionary.

You are allowed to **add tuples to tuples (or multiply a tuple)**, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple.

▼ Let's create a tuple

```
a = tuple('Hello, world!')  
b = ('hello', 'world')  
c = 'hello', 'world'  
print(a)  
print(b)  
print(c)  
print(type(a))  
print(type(b))  
print(type(c))
```

→

```
('H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')  
('hello', 'world')  
('hello', 'world')  
<class 'tuple'>  
<class 'tuple'>  
<class 'tuple'>
```

▼ Operations with tuples

```
b = ('hello', 'world')  
print(b[0])
```

```
b[0] = 'new'
```

→

hello

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3, in <module>

```
b[0] = 'new'
```

TypeError: 'tuple' object does not support item assignment

(we can take an element from a tuple, but not change or delete it; still we can delete a whole tuple)

```
b = ['hello', 'world']
print(tuple(b))
print(type(tuple(b)))
```

→

('hello', 'world')

<class 'tuple'>

```
b = ('hello', 'world')
print(list(b))
print(type(list(b)))
print(b)
print(type(b))
```

→

['hello', 'world']

<class 'list'>

('hello', 'world')

<class 'tuple'>

```
b = ('hello', 'world')
a = list(b)
a[0] = 'new'
print(tuple(a))
```

```
print(type(tuple(a)))
print(type(a))
```

→

```
('new', 'world')
<class 'tuple'>
<class 'list'>
```

▼ Tuple methods

- ▼ tuple.count() - returns a number of times a specified value occurs in a tuple

```
b = ('hello', 'world', 'hello')
print(b.count('hello'))
print(b.index('world'))
```

→

```
2
1
```

- ▼ tuple.index() - searches the tuple for a specified value and returns its position

```
b = ('hello', 'world', 'hello')
print(b.count('hello'))
print(b.index('world'))
```

→

```
2
1
```

▼ Unpacking a Tuple

In Python, we are also allowed to extract the values back into variables. This is called "unpacking":

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
```

```
print(green)
print(yellow)
print(red)
```

→

apple
banana
cherry

Note: *the number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.*

▼ Loop Tuples

▼ Loop through a tuple

```
butterflies = ("monarch", "pharaoh", "papillon")
for x in butterflies:
    print(x)
```

→

monarch
pharaoh
papillon

▼ Loop by index

```
butterflies = ("monarch", "pharaoh", "papillon")
for x in range(len(butterflies)):
    print(butterflies[x])
```

→

monarch
pharaoh
papillon

▼ While Loop

```
butterflies = ("monarch", "pharaoh", "papillon")
i = 0
while i < len(butterflies):
    print(butterflies[i])
    i = i+1
```

→

monarch
pharaoh
papillon

▼ Set (set) - an unordered collection of unique objects.

Set items are unchangeable, but you can remove items and add new items. Also, set items are unindexed, here's why there's no duplicate values in a set.

▼ Let's make a set

```
a = set()
print(a)
b = set("Morning!")
print(b)
c = {1, 2, 3, 4}
print(c)
d = {i for i in range(10)}
print(d)
e = {}
print(type(e))
```

→

{'o', 'n', 'M', 'i', 'g', '!', 'r'}
{1, 2, 3, 4}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
<class 'dict'>

▼ Access items

You cannot access items in a set by referring to an index or a key. But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.

▼ for loop

```
butterflies = {"monarch", "pharaoh", "papillon"}  
for x in butterflies:  
    print(x)
```

→

pharaoh
monarch
papillon

▼ in keyword

```
butterflies = {"monarch", "pharaoh", "papillon"}  
print("pharaoh" in butterflies)
```

→

True

▼ Loop set

```
colors = {'red', 'green', 'blue', 'pink'}  
for x in colors:  
    print(x)
```

→

blue
green
red
pink

Sets are handy when we need to get rid of repetitions:

```
a = ['cat', 'cat', 'mouse']  
print(a)  
print(set(a))
```

→

```
['cat', 'cat', 'mouse']
{'mouse', 'cat'}
```

frozenset - like set, but immutable

▼ Set Methods:

- ▼ a.union(b) - returns a set that is a merge of a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'fish', 'turtle', 'hamster'}
c = a.union(b)
print(c)
```

→

```
{'fish', 'hamster', 'cat', 'mouse', 'dog', 'turtle'}
```

- ▼ a.update(b) - adds elements from b to a (not only set, any iterable)

```
a = {'cat', 'dog', 'mouse'}
b = {'fish', 'turtle', 'hamster'}
a.update(b)
print(a)
```

→

```
{'dog', 'fish', 'mouse', 'cat', 'hamster', 'turtle'}
```

- ▼ a.intersection(b) - returns a set that is an intersection of a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.intersection(b)
print(c)
```

→

```
{'cat'}
```

- ▼ `a.intersection_update(b)` - leaves in the set a only those elements that are present in the set b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.intersection_update(b)
print(a)
```

→

{'cat'}

- ▼ `a.difference(b)` - returns the difference between a and b (those elements that are present in a but not in b)

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.difference(b)
print(c)
```

→

{'dog', 'mouse'}

- ▼ `a.difference_update(b)` - deletes from a elements from b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.difference_update(b)
print(a)
```

→

{'mouse', 'dog'}

- ▼ `a.symmetric_difference(b)` - returns symmetric difference between a and b (elements, present in a or b, but not in both)

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.symmetric_difference(b)
print(c)
```

→

```
{'hamster', 'dog', 'turtle', 'mouse'}
```

- ▼ `a.symmetric_difference_update(b)` - puts in a symmetric difference between a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.symmetric_difference_update(b)
print(a)
```

→

```
{'hamster', 'dog', 'mouse', 'turtle'}
```

- ▼ `a.issubset(b)` - returns True if a is a subset of b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
print(a.issubset(b))
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(c.issubset(d))
```

→

```
False
```

```
True
```

- ▼ `a.issuperset(b)` - returns True if b is a subset of a

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(d.issuperset(c))
```

→

```
True
```

- ▼ `a<b` - equal to $a \leq b$ and $a \neq b$

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(c<d)
```

→

True

- ▼ a>b - equal to a≥b and a≠b

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(d>c)
```

→

True

- ▼ set.add() - adds an element to the set

```
c = {'cat'}
c.add('turtle')
print(c)
```

→

{'turtle', 'cat'}

- ▼ set.clear() - removes all elements from the set

```
s = {'mouse', 'cat', 'house'}
print(s)
s.clear()
print(s)
```

→

{'mouse', 'house', 'cat'}
set()

- ▼ set.copy() - returns a copy of the set

```
s = {'mouse', 'cat', 'house'}
print(s.copy())
d = s.copy()
print(d)
```

→

```
{'mouse', 'house', 'cat'}
{'mouse', 'house', 'cat'}
```

▼ **set.discard()** - removes the specified element

```
s = {'mouse', 'cat', 'house'}
s.discard('mouse')
print(s)
```

→

```
{'house', 'cat'}
```

▼ **set.isdisjoint()** - returns whether two sets have an intersection (True if nothing in common)

```
s = {'mouse', 'cat', 'house'}
s1 = {'mouse', 'cat', 'castle'}
print(s1.isdisjoint(s))
s2 = {'house'}
s3 = {'mouse', 'cat', 'castle'}
print(s2.isdisjoint(s3))
```

→

False

True

▼ **set.pop()** - removes a random element from the set

```
s = {'mouse', 'cat', 'house'}
s.pop()
print(s)
s.pop()
print(s)
```

```
s.pop()  
print(s)
```

→

```
{'cat', 'mouse'}  
{'mouse'}  
set()
```

▼ **set.remove()** - removes a specified element from the set

```
s = {'mouse', 'cat', 'house'}  
s.remove('mouse')  
print(s)
```

→

```
{'cat', 'house'}
```

▼ **Dictionary** (dict) - unoredered collections of various values that have keys. One can get access to a value by using a key, and vice versa. Dict are also called “associative massives” or “hash-tables”. No duplicates (no two items with the same key).

▼ Let's create a dict

```
d = {} #using a literal  
print(d)  
d1 = {1:"Moon", 2:"Earth"} #using a literal  
print(d1)  
d2 = dict(race='elf', weapon='magic') #using dict function  
print(d2)  
d3 = dict([(1,'dollar'),(2,'ruble')]) #using a literal  
print(d3)  
d4 = dict.fromkeys(['b', 'a'], 100) #using fromkeys method  
print(d4)  
d5 = {x: x**3 for x in range(7)} #using a dictionary generator  
print(d5)
```

→

```
{}  
{1: 'Moon', 2: 'Earth'}
```

```
{'race': 'elf', 'weapon': 'magic'}  
{1: 'dollar', 2: 'ruble'}  
  
{'b': 100, 'a': 100}  
  
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216}
```

▼ Add items

```
thisdict = {  
    "brand": "Gucci",  
    "item": "bag",  
    "storage": 3  
}  
thisdict["color"] = "red"  
print(thisdict)
```

→

```
{"brand": 'Gucci', 'item': 'bag', 'storage': 3, 'color': 'red'}
```

▼ Remove items

Apart from `.pop()` and `.popitem()`, you can use `del` to delete a specific item from a dict:

```
thisdict = {  
    "brand": "Gucci",  
    "item": "bag",  
    "storage": 3  
}  
del thisdict["storage"]  
print(thisdict)
```

→

```
{"brand": 'Gucci', 'item': 'bag'}
```

`del` can also delete the whole dict

▼ Let's get a value and a key from a dict, and change values

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1[1])  
d1[2] = 'Mars'  
print(d1)
```

→

Moon

{1: 'Moon', 2: 'Mars'}

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1['Moon'])
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 2, in <module>

print(d1['Moon'])

KeyError: 'Moon'

▼ Loop a dict

▼ get keys

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",  
    "le chat": "a cat",  
    "le soir": "the evening"  
}  
  
for x in French_vocab:  
    print(x)
```

→

le fromage

le lait

le chat

le soir

▼ get values

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",
```

```
"le chat":"a cat",
"le soir":"the evening"
}

for x in French_vocab:
    print(French_vocab[x])
```

→

cheese
milk
a cat
the evening

▼ .values()

```
French_vocab = {
    "le fromage":"cheese",
    "le lait":"milk",
    "le chat":"a cat",
    "le soir":"the evening"
}

for x in French_vocab.values():
    print(x)
```

→

cheese
milk
a cat
the evening

▼ .keys()

```
French_vocab = {
    "le fromage":"cheese",
    "le lait":"milk",
    "le chat":"a cat",
    "le soir":"the evening"
}

for x in French_vocab.keys():
    print(x)
```

→

le fromage
le lait
le chat
le soir

▼ .items()

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",  
    "le chat": "a cat",  
    "le soir": "the evening"  
}  
  
for x, y in French_vocab.items():  
    print(x, y)
```

→

le fromage cheese
le lait milk
le chat a cat
le soir the evening

▼ Copy a dict

Use .copy() or built-in func. dict()

▼ Nested dict

A dictionary can contain dictionaries, this is called nested dictionaries.

```
French_vocab = {  
    "food" : {  
        "le fromage": "cheese",  
        "le lait": "milk"  
    },  
    "animals" : {  
        "le chat": "a cat",  
        "l'elephant": "an elephant"  
    },  
    "time" : {  
        "le soir": "tonight",  
        "le matin": "this morning"
```

```

        }
    }

print(French_vocab)
print(French_vocab["time"])
for x in French_vocab:
    print(x)
for x in French_vocab["animals"].values():
    print(x)

```

→

```

{'food': {'le fromage': 'cheese', 'le lait': 'milk'}, 'animals': {'le chat': 'a cat', "l'elephant": "an elephant"}, 'time': {'le soir': 'tonight', 'le matin': 'this morning'}}}
{'le soir': 'tonight', 'le matin': 'this morning'}
food
animals
time
a cat
an elephant

```

▼ Dict Methods

▼ dict.clear() - clears up the dict

```

d1 = {1:"Moon", 2:"Earth"}
print(d1)
d1.clear()
print(d1)

```

→

```

{1: 'Moon', 2: 'Earth'}
{}

```

▼ dict.copy() - returns a copy of the dict

```

d1 = {1:"Moon", 2:"Earth"}
d2 = d1.copy()
print(d2)
print(d1)

```

→

```
{1: 'Moon', 2: 'Earth'}  
{1: 'Moon', 2: 'Earth'}
```

- ▼ dict.get(key, [default]) - returns the value of the key, if none - returns default (None)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.get(1))  
print(d1.get("Mars"))  
print(d1.get(3))
```

→

```
Moon  
None  
None
```

- ▼ dict.items() - returns key - value pairs that are in the dict

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.items())
```

→

```
dict_items([(1, 'Moon'), (2, 'Earth')])
```

- ▼ dict.keys() - returns keys

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.keys())
```

→

```
dict_keys([1, 2])
```

- ▼ dict.pop(key [, default]) - deletes the key and returns the value, if none - returns default (exception)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1)
```

```
print(d1.pop(1))
print(d1)
```

→

{1: 'Moon', 2: 'Earth'}

Moon

{2: 'Earth'}

- ▼ dict.popitem() - deletes and returns key-value pair, if {} - KeyError (no order)

```
d1 = {1:"Moon", 2:"Earth"}
print(d1)
print(d1.popitem())
print(d1)
print(d1.popitem())
print(d1)
print(d1.popitem())
print(d1)
```

→

{1: 'Moon', 2: 'Earth'}

(2, 'Earth')

{1: 'Moon'}

(1, 'Moon')

{}

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 7, in <module>

print(d1.popitem())

KeyError: 'popitem(): dictionary is empty'

- ▼ dict.setdefault(key[, default]) - returns the value of the key, if none - creates a key with default value (None)

```
d1 = {1:"Moon", 2:"Earth"}
print(d1.setdefault(1))
print(d1.setdefault(3))
```

→

Moon

None

- ▼ `dict.update([other])` - updates the dict by creating key-value pairs from [other]. Existing keys change. Returns None (not a new dict)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1)  
d1.update({3:"Venus"})  
print(d1)  
d1.update({3:"Mars"})  
print(d1)
```

→

```
{1: 'Moon', 2: 'Earth'}  
{1: 'Moon', 2: 'Earth', 3: 'Venus'}  
{1: 'Moon', 2: 'Earth', 3: 'Mars'}
```

- ▼ `dict.values()` - returns values

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.values())
```

→

```
dict_values(['Moon', 'Earth'])
```

Classes and Objects

A **class** is like an object constructor, “a blueprint” for creating new objects (or subclasses). A class has features (attributes) and actions (methods) that objects of this class (or subclasses) inherit. A class can be created during runtime, and changed anytime.

An **object** - basically anything in Python (and everything has its class, hidden in the core of the language).

An object has:

- a state - represented by attributes, reflects the properties of an object;

- some behaviour - represented by methods of an object, reflects the response of an object to other objects;
- an identity - object is given a unique name which enables one object to interact with other objects.

When an object is created (declared), we say that a class has been instantiated.

▼ Let's create a class

```
class MyClass:  
    x = 5  
  
print(MyClass)
```

→

<class 'main.MyClass'>

▼ Let's create an object

```
class MyClass:  
    x = 5  
  
print(MyClass)  
  
p1 = MyClass()  
print(p1.x)
```

→

<class 'main.MyClass'>

5

▼ The `__init__()` Function

Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created

▼ Example

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
person1 = Person("Julie", 16)  
  
print(person1.name)  
print(person1.age)
```

→

Julie

16

The `__init__()` function is called automatically every time the class is being used to create a new object.

▼ Object Methods

Methods in objects are functions that belong to the object.

▼ Example

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def my_name_is(self):  
        print("Hello, my name is " + self.name + ".")  
  
    def im_age(self):  
        print("I'm " + str(self.age) + " years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()
```

▼ The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong to the class. It does not have to be named `self`, you can call it whatever you like, but it has to be the first parameter of any function in the class.

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(per):  
        print("Hello, my name is " + per.name + ".")  
  
    def im_age(per):  
        print("I'm " + str(per.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()
```

→

Hello, my name is Julie.

I'm 16 years old.

▼ Modify Object Properties

You can modify properties on objects like this:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(person1):  
        print("Hello, my name is " + person1.name + ".")  
  
    def im_age(person1):  
        print("I'm " + str(person1.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()  
  
person1.age = 17  
print((person1.age))
```

→

Hello, my name is Julie.

I'm 16 years old.

17

▼ Delete Object Properties

You can delete properties on objects by using the `del` keyword:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(person1):  
        print("Hello, my name is " + person1.name + ".")  
  
    def im_age(person1):  
        print("I'm " + str(person1.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()  
  
del person1.age  
print((person1.age))
```

→

Hello, my name is Julie.

I'm 16 years old.

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 17, in <module>

print((person1.age))

AttributeError: 'Person' object has no attribute 'age'

▼ Delete Objects

You can delete objects by using the `del` keyword:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):
```

```

some_self.name = name
some_self.age = age

def my_name_is(person1):
    print("Hello, my name is " + person1.name + ".") 

def im_age(person1):
    print("I'm " + str(person1.age) +" years old.")

person1 = Person("Julie", 16)
person1.my_name_is()
person1.im_age()

del person1

print(person1)

```

→

Hello, my name is Julie.

I'm 16 years old.

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 18, in <module>

print((person1))

NameError: name 'person1' is not defined

▼ The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the `pass` statement to avoid getting an error.

▼ Example

```

class Person:
    pass
print(Person)

```

→

<class 'main.Person'>

```

class Alien:
    pass
print(Alien)

```

→

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3

print(Alien)

^

IndentationError: expected an indented block

Class Methods:

- ▼ classmethod() - returns a class method for a function
- ▼ delattr() - deletes an attribute from an object
- ▼ getattr() - returns the value of a named attribute of an object
- ▼ hasattr() - returns True if an object has a given attribute
- ▼ isinstance() - determines whether an object is an instance of a given class
- ▼ issubclass() - determines whether an object is an instance of a given subclass
- ▼ property() - returns a property value of a class
- ▼ setattr() - sets the value of a named attribute of an object
- ▼ super() - returns a proxy object that delegates method calls to parent or sibling class

Input/ Output

- ▼ format() - converts a value to a formatted representation
- ▼ input() - reads input from the console
- ▼ open() - opens a file and returns a file object
- ▼ print() - prints to a text stream or the console

Variables, References, and Scope

- ▼ dir() - returns a list of names in current local scope or a list of object attributes
- ▼ globals() - returns a dictionary representing the current global symbol table
- ▼ id() - returns the identity of an object
- ▼ locals() - updates and returns a dictionary representing current local symbol table

- ▼ `vars()` - returns `_dict_` attribute for a module, class, or object

Miscellaneous

- ▼ `callable()` - returns `True` if object appears callable
- ▼ `compile()` - compiles source into a code or AST object
- ▼ `eval()` - evaluates a Python expression
- ▼ `exec()` - implements dynamic execution of Python code
- ▼ `hash()` - returns the hash value of an object
- ▼ `help()` - invokes the built-in help system
- ▼ `memoryview()` - returns a memory view object
- ▼ `staticmethod()` - returns a static method for a function
- ▼ `__import__()` - invoked by the `import` statement

Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class inherits from base class, and child class - from parent class.

- ▼ Example

```
class Person:  
    def __init__(self, fname, lname, gender):  
        self.firstname = fname  
        self.lastname = lname  
        self.gender = gender  
  
    def greeting(self):  
        print("Hello! My name is "+self.firstname+" "+self.lastname+".")  
  
class Student(Person):  
    def __init__(self, fname, lname, gender, faculty):  
        Person.__init__(self, fname, lname, gender)  
        self.university = "Harvard"  
        self.faculty = faculty  
  
    def myuniversity(self):  
        print("I study in "+self.university+".")
```

```

def myfaculty(self):
    print("I study at the "+self.faculty+" faculty.")

def accomodation(self):
    if self.gender == "male":
        print("I live in the male dormitory.")
    else:
        print("I live in the female dormitory.")

print("Here's a man who looks like being in his forties:\n")
y = Person("Finn", "Olleyson", "male")
y.greeting()
print()
print("Here's a girl who looks like being in her twenties:\n")
x = Student("Emma", "Tompson", "female", "Computer Science")
x.greeting()
x.myuniversity()
x.myfaculty()
x.accomodation()

```

→

Here's a man who looks like being in his forties:

Hello! My name is Finn Olleyson.

Here's a girl who looks like being in her twenties:

Hello! My name is Emma Tompson.

I study in Harvard.

I study at the Computer Science faculty.

I live in the female dormitory.

The child's `__init__()` function **overrides** the inheritance of the parent's `__init__()` function.

To keep the inheritance of the parent's `__init__()` function, add a call to the parent's `__init__()` function.

`super()` function makes the child class inherit all the methods and properties from its parent without naming the parent class.

Iterators

An iterator is an object that contains a countable number of values. It can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods `__iter__()` and `__next__()`.

Iterator vs Iterable

Lists, tuples, dictionaries, and sets (and str) are all iterable objects. They are iterable *containers* which you can get an iterator from.

- ▼ All these objects have a `iter()` method which is used to get an iterator:

```
tup = ("tiger", "lion", "panther")
myit = iter(tup)

print(next(myit))
print(next(myit))
print(next(myit))
```

→

tiger
lion
panther

- ▼ Loop

```
tup = ("tiger", "lion", "panther")
for x in tup:
    print(x)
```

→

tiger
lion
panther

- ▼ Class Iterator

To create an object/class as an iterator you have to implement the methods `__iter__()` and `__next__()` to your object.

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
```

```

        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))

```

→

1
2
3
4
5

▼ StopIteration

To prevent the iteration to go on forever, we can use the `StopIteration` statement.

In the `__next__()` method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

```

class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        if self.a <= 20:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

```

```
for x in myiter:  
    print(x)
```

→

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Sope

“Уровень вложенности” in Russian. It can be local or global.

Modules

Consider a module to be the same as a code library. A file containing definition of functions, classes, variables, constants or any other Python object, even some executable code you want to include in your application.

▼ Create a module

To create a module just save the code you want in a file with the file extension `.py`

▼ Use a module

Use the `import` statement (import a module). When using a function from a module, use the syntax: *module_name.function_name*

▼ Variables in a module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc).

▼ Naming and renaming

You can use any name.py. You can create an alias when you import a module, by using the `as` keyword (import some_module as sm).

▼ Built-in modules

To see the whole list print: `help('modules')`. Below there are some frequently used modules.

▼ os module

Has functions to perform many tasks of operating system.

▼ `mkdir()` - creates a new directory.

A new directory corresponding to path in string argument in the function will be created. If we open D drive in Windows explorer we should notice tempdir folder created.

```
>>> import os  
>>> os.mkdir("d:\\tempdir")
```

▼ `chdir()`- changes current working directory

```
>>> import os  
>>> os.chdir("d:\\temp")
```

▼ `getcwd()` - returns name of a current working directory

```
>>> os.getcwd()  
'd:\\temp'
```

Directory paths can also be relative. If current directory is set to D drive and then to temp without mentioning preceding path, then also current working directory will be changed to d:\temp.

```
>>> os.chdir("d:\\\\")
>>> os.getcwd()
'd:\\\\'
>>> os.chdir("temp")
>>> os.getcwd()
'd:\\\\temp'
```

In order to set current directory to parent directory use ".." as the argument to chdir() function.

```
>>> os.chdir("d:\\\\temp")
>>> os.getcwd()
'd:\\\\temp'
>>> os.chdir("..")
>>> os.getcwd()
'd:\\\\'
```

- ▼ rmdir() - removes a specific directory either with absolute or relative path (it shouldn't be current working directory, an it shouldn't be empty)

```
>>> os.chdir("tempdir")
>>> os.getcwd()
'd:\\\\tempdir'
>>> os.rmdir("d:\\\\temp")
PermissionError: [WinError 32] The process cannot access the file because it
is being used by another process: 'd:\\\\temp'
>>> os.chdir("..")
>>> os.rmdir("temp")
```

- ▼ listdir() - returns a list of all files in a directory

```
>>> os.listdir("c:\\\\Users")
['acer', 'All Users', 'Default', 'Default User', 'desktop.ini', 'Public']
```

- ▼ random module

Contains randomization functions.

Python uses a pseudo-random generator based upon Mersenne Twister algorithm that produces 53-bit precision floats. Functions in this module depend on pseudo-random number generator function `random()` which generates a random float number between 0.0 and 1.0.

- ▼ `random.random` - returns a random float number between 0.0 and 1.0 (no args).

```
>>> import random  
>>> random.random()  
0.755173688207591
```

- ▼ `random.randint()` - returns a random number between the given numbers

```
>>> random.randint(1,100)  
91
```

- ▼ `random.randrange()` - returns a random element from the range (start, stop, step args)

```
>>> random.randrange(0,101,10)  
40
```

- ▼ `random.choice()` - returns a randomly selected eleemnt from a sequence object (e.g. str, list, tuple).If empty - IndexError.

```
>>> import random  
>>> random.choice('computer')  
'o'
```

- ▼ `random.shuffle()` - randomly reorders elements in a list

```
>>> numbers=[12,23,45,67,65,43]  
>>> random.shuffle(numbers)  
>>> numbers  
[23, 12, 43, 65, 67, 45]
```

▼ math module

This module presents commonly required mathematical functions (trigonometric, representation, logarithmic, angle conversion functions).

In addition, two mathematical constants are also defined in this module (Pie, Euler's number, etc.).

▼ Trigonometric functions

▼ radians() - converts angle in degrees to radians

```
>>> math.radians(30)
0.5235987755982988
```

▼ degrees()- converts angle in radians in degrees

```
>>> math.degrees(math.pi/6)
29.99999999999996
```

▼ math.log() - returns natural logarithm of given number; natural logarithm is calculated on the base e

math.log10(): returns base-10 logarithm or standard logarithm of given number.

```
>>> math.log10(10)
1.0
```

▼ math.exp() - returns a float number after raising e to given number : exp(x) is equivalent to e**x

```
>>> math.e**10
22026.465794806703
```

▼ math.pow() - receives two args, raises first to the second, returns the result

```
>>> math.pow(4,4)  
256.0
```

▼ `math.sqrt()` - computes square root of given number

```
>>> math.sqrt(100)  
10.0
```

▼ Representation functions

▼ `math.ceil()` - approximates the given number to the smallest integer greater than or equal to the given float number

```
>>> math.ceil(4.5867)  
5
```

▼ `math.floor()` - returns the largest integer less than or equal to the given number

```
>>> math.floor(4.5687)  
4
```

▼ time module

▼ `time()` - returns current system time in ticks. The ticks is number of seconds elapsed after epoch time i.e. 12.00 am, January 1, 1970.

```
>>> time.time()  
1544348359.1183174
```

▼ `localtime()` - translates time in ticks in a time tuple notation.

```
>>> tk=time.time()  
>>> time.localtime(tk)  
time.struct_time(tm_year=2018, tm_mon=12, tm_mday=9, tm_hour=15, tm_min=11, tm_sec=25, tm_wday=6, tm_yday=343, tm_isdst=0)
```

- ▼ `asctime()` - returns a readable format of local time

```
>>> tk=time.time()  
>>> tp=time.localtime(tk)  
>>> time.asctime(tp)  
'Sun Dec  9 15:11:25 2018'
```

- ▼ `ctime()` - returns string representation of system's current time

```
>>> time.ctime()  
'Sun Dec  9 15:17:40 2018'
```

- ▼ `sleep()` - halts current program execution for a specified duration in seconds

```
>>> time.ctime()  
'Sun Dec  9 15:19:14 2018'  
>>> time.sleep(20)  
>>> time.ctime()  
'Sun Dec  9 15:19:34 2018'
```

▼ sys module

Provides functions and variables used to manipulate different parts of the Python runtime environment.

- ▼ `sys.argv` - returns the list of command line args passed to a Python script Item[0] - script name. Rest of the arguments are stored at subsequent indices.

Here is a Python script (`test.py`) consuming two arguments from command line.

```
import sys  
print ("My name is {}. I am {} years old".format(sys.argv[1], sys.argv[2]))
```

This script is executed from command line as follows:

```
C:\python37>python tmp.py Anil 23  
My name is Anil. I am 23 years old
```

▼ sys.exit - causes program to end and return to either Python console or command prompt. It is used to safely exit from program in case of exception.

▼ sys.maxsize - returns the largest integer a variable can take

```
>>> import sys  
>>> sys.maxsize  
9223372036854775807
```

▼ sys.path - an environment variable that returns search path for all Python modules

```
>>> sys.path  
['', 'C:\\\\python37\\\\Lib\\\\idlelib', 'C:\\\\python37\\\\python36.zip', 'C:\\\\python3  
7\\\\DLLs', 'C:\\\\python37\\\\lib', 'C:\\\\python37', 'C:\\\\Users\\\\acer\\\\AppData\\\\Ro  
aming\\\\Python\\\\Python37\\\\site-packages', 'C:\\\\python37\\\\lib\\\\site-packages']
```

▼ sys.stdin , sys.stdout , sys.stderr - file objects used by the interpreter for standard input, output and errors. stdin is used for all interactive input (Python shell). stdout is used for the output of print() and of input(). The interpreter's prompts and error messages go to stderr.

▼ sys.version - displays a string containing version number of current Python interpreter

▼ collections module

Provides alternatives to built-in container data types such as list, tuple and dict.

▼ namedtuple() - a factory function that returns object of a tuple subclass with named fields. Any valid Python identifier may be used for a field name except for names starting with an underscore.

```
collections.namedtuple(typename, field-list)
```

The typename parameter is the subclass of tuple. Its object has attributes mentioned in field list. These field attributes can be accessed by lookup as well as by its index.

Following statement declares a employee namedtuple having name, age and salary as fields

```
>>> import collections  
>>> employee=collections.namedtuple('employee', [name, age, salary])  
To create a new object of this namedtuple  
>>> e1=employee("Ravi", 251, 20000)
```

Values of the field can be accessible by attribute lookup

```
>>> e1.name  
'Ravi'
```

Or by index

```
>>> e1[0]  
'Ravi'
```

▼ `OrderedDict()` - Ordered dictionary is similar to a normal dictionary. However, normal dictionary the order of insertion of keys in it whereas ordered dictionary object remembers the same. The key-value pairs in normal dictionary object appear in arbitrary order.

```
>>> d1={}
>>> d1['A']=20
>>> d1['B']=30
>>> d1['C']=40
>>> d1['D']=50
```

We then traverse the dictionary by a for loop,

```
>>> for k,v in d1.items():
print (k,v)

A 20
B 30
D 50
C 40
```

But in case of `OrderedDict` object:

```
>>> import collections  
>>> d2=collections.OrderedDict()  
>>> d2['A']=20  
>>> d2['B']=30  
>>> d2['C']=40  
>>> d2['D']=50
```

Key-value pairs will appear in the order of their insertion.

```
>>> for k,v in d2.items():  
    print (k,v)  
A 20  
B 30  
C 40  
D 50
```

▼ deque() - A deque object supports append and pop operation from both ends of a list. It is more memory efficient than a normal list object because in a normal list, removing one of item causes all items to its right to be shifted towards left. Hence it is very slow.

```
>>> q=collections.deque([10,20,30,40])  
>>> q.appendleft(110)  
>>> q  
deque([110, 10, 20, 30, 40])  
>>> q.append(41)  
>>> q  
deque([0, 10, 20, 30, 40, 41])  
>>> q.pop()  
40  
>>> q  
deque([0, 10, 20, 30, 40])  
>>> q.popleft()  
110  
>>> q  
deque([10, 20, 30, 40])
```

▼ statistics module

▼ mean() - calculate arithmetic mean of numbers in a list

```
>>> import statistics  
>>> statistics.mean([2, 5, 6, 9])  
5.5
```

- ▼ median() - returns middle value of numeric data in a list. For odd items in list, it returns value at $(n+1)/2$ position. For even values, average of values at $n/2$ and $(n/2)+1$ positions is returned.

```
>>> import statistics  
>>> statistics.median([1, 2, 3, 8, 9])  
3  
>>> statistics.median([1, 2, 3, 7, 8, 9])  
5.0
```

- ▼ mode() - returns most repeated data point in the list

```
>>> import statistics  
>>> statistics.mode([2, 5, 3, 2, 8, 3, 9, 4, 2, 5, 6])  
2
```

- ▼ stdev() - calculates standard deviation on given sample in the form of list

```
>>> import statistics  
>>> statistics.stdev([1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5])  
1.3693063937629153
```

Some more modules ([TBA](#)):

- ▼ re module ([TBA](#))

Модуль для работы с регулярными выражениями

- ▼ match() - ищет последовательность в начале строки
- ▼ search() - ищет первое совпадение с шаблоном
- ▼ findall() - ищет все совпадения с шаблоном, возвращает результирующие строки в виде списка
- ▼ finditer() - ищет все совпадения с шаблоном, возвращает итератор

- ▼ `compile()` - компилирует регуляторное выражение (к этому объекту затем можно применять все перечисляемые функции)
 - ▼ `fullmatch()` - вся строка должна соответствовать описанному регулярному выражению
 - ▼ `re.sub()` - для замены в строках
 - ▼ `re.split()` - для разделения строки на части
- ▼ **threading module (TBA)**

Thread-based parallelism. This module constructs higher-level threading interfaces on top of the lower level `_tread` module.

- ▼ `active_count()` - return the number of Thread objects currently alive (the returned count is equal to the length of the list returned by `enumerate()`).
- ▼ `current_thread()` - return the current `Thread` object, corresponding to the caller's thread of control. If the caller's thread of control was not created through the `threading` module, a dummy thread object with limited functionality is returned.
- ▼ `excepthook()` - handle uncaught exception raised by `Thread.run()`.

The `args` argument has the following attributes:

- `exc_type`: Exception type.
 - `exc_value`: Exception value, can be `None`.
 - `exc_traceback`: Exception traceback, can be `None`.
 - `thread`: Thread which raised the exception, can be `None`.
- ▼ `__excepthook__` - holds the original value of `threading.excepthook()`. It is saved so that the original value can be restored in case they happen to get replaced with broken or alternative objects.
- ▼ `get_ident()` - return the ‘thread identifier’ of the current thread.
- ▼ `get_native_id()` - return the native integral Thread ID of the current thread assigned by the kernel
- ▼ `enumerate()` - return a list of all `Thread` objects currently active
- ▼ `main_thread()` - Return the main `Thread` object.

- ▼ `settrace(func)` - set a trace function for all threads started from the `threading` module
- ▼ `gettrace()` - get the trace function as set by `settrace()`
- ▼ `setprofile(func)` - Set a profile function for all threads started from the `threading` module
- ▼ `getprofile()` - get the profiler function as set by `setprofile()`
- ▼ `stack_size([size])` - return the thread stack size used when creating new threads
- ▼ `TIMEOUT_MAX()` - The maximum value allowed for the *timeout* parameter of blocking functions (`Lock.acquire()`, `RLock.acquire()`, `Condition.wait()`, etc.)
- ▼ tkinter module
- ▼ csv module
- ▼ pickle module
- ▼ socket module
- ▼ sqlite3 module
- ▼ json module
- ▼ itertools module (все они создают итераторы)
 - ▼ `count(start=, step=)` - returns an iterator of evenly spaced values (infinite iterator)

```
import itertools
x = itertools.count(start=10, step=5)
print(next(x))
print(next(x))
print(next(x))
print(next(x))
```

→

10, 15, 20, 25

```
import itertools
x = itertools.count(start=10, step=5)
for i in x:
    if i>35:
```

```
        break
    else:
        print(i, end=" ")
```

→

10 15 20 25 30 35

```
import itertools
l1 = [1, 2, 3, 4, 5]
l2 = zip(itertools.count(0, 1), l1)
print(list(l2))
```

→

[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]

```
import itertools
l1 = map(lambda x: x**2, itertools.count(0, 1))
for i in l1:
    if i > 60:
        break
    else:
        print(i, end=" ")
```

→

0 1 4 9 16 25 36 49

- ▼ `cycle(obj)` - returns each element from given iterable and saves its copy, when iterating object ends, returns copy - this repetition forms an infinite loop (infinite iterator)

```
import itertools
l1 = [1, 2, 3]
l2 = itertools.cycle(l1)
count = 0
for i in l2:
    if count > 15:
        break
    else:
        print(i, end=" ")
    count += 1
```

→

1 2 3 1 2 3 1 2 3 1 2 3 1

```
import itertools
s = "Go! "
i = itertools.cycle(s)
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
```

→

G
o
!

G
o
!

- ▼ `repeat(obj, times)` - returns the object argument repeatedly (infinite iterator)

```
import itertools
s = "Go! "
i = itertools.repeat(s)
print(next(i))
print(next(i))
print(next(i))
```

→

Go!
Go!
Go!

```
import itertools
iterobj = itertools.repeat("Go!", times=10)
```

```
for i in iterobj:  
    print(i, end=" ")
```

→

Go! Go! Go! Go! Go! Go! Go! Go! Go!

```
import itertools  
powobj = map(pow, range(15), itertools.repeat(2))  
for i in powobj:  
    print(i, end=" ")
```

→

0 1 4 9 16 25 36 49 64 81 100 121 144 169 196

```
import itertools  
l = [1, 2, 3]  
obj = zip(itertools.repeat(5), l)  
print(list(obj))  
obj2 = zip(l, itertools.repeat(7))  
print(list(obj2))
```

→

[(5, 1), (5, 2), (5, 3)]

[(1, 7), (2, 7), (3, 7)]

```
import itertools  
l = ['Lolly', 'Molly', 'Polly']  
obj = zip(itertools.repeat("Hello"), l)  
print(next(obj))  
print(next(obj))  
print(next(obj))
```

→

('Hello', 'Lolly')

('Hello', 'Molly')

('Hello', 'Polly')

- ▼ `accumulate(iterable,func,*initial=None)` - applies a function to successive items in a list (finite iterator)

`functools.reduce()` возвращает только конечное накопленное значение для аналогичной функции.

```
import itertools
#using add and mul operator,so importing operator module
import operator
#using reduce(),so importing reduce() from functools module
from functools import reduce

l1=itertools.accumulate([1,2,3,4,5])
print(list(l1))
```

→

[1, 3, 6, 10, 15]

```
r1=reduce(operator.add,[1,2,3,4,5])
print (r1)
```

→

15

```
l2=itertools.accumulate([1,2,3,4,5],operator.add,initial=10)
print (list(l2))
```

→

[10, 11, 13, 16, 20, 25]

```
l3=itertools.accumulate([1,2,3,5,5],operator.mul)
print (list(l3))
```

→

[1, 2, 6, 30, 150]

```
r2=reduce(operator.mul,[1,2,3,4,5])  
print(r2)
```

→

120

```
l4=itertools.accumulate([2,4,6,3,1],max)  
print (list(l4))
```

→

[2, 4, 6, 6, 6]

```
r3=reduce(max,[2,4,6,3,1])  
print (r3)
```

→

6

```
l5=itertools.accumulate([2,4,6,3,1],min)  
print(list(l5))
```

→

[2, 2, 2, 2, 1]

```
r4=reduce(min,[2,4,6,3,1])  
print (r4)
```

→

1

- ▼ `chain()` - (finite iterator) создает итератор, который возвращает элемент из итерируемого объекта до тех пор, пока он не закончится, а потом переходит к

следующему. Он будет рассматривать последовательности, идущие друг за другом, как одну.(finite iterator)

```
import itertools

l1 = itertools.chain(["cat", "dog"], [5, 7, 9], "LOL")
print(list(l1))
```

→

```
['cat', 'dog', 5, 7, 9, 'L', 'O', 'L']
```

▼ `chain.from_iterable(iterable)` - берет один итерируемый объект в качестве входного аргумента и возвращает «склеенный» итерируемый объект, содержащий все элементы входного (finite iterator)

```
import itertools

l1 = itertools.chain.from_iterable(["kot", "kot", "kot"])
print((list(l1)))
```

→

```
['k', 'o', 't', 'k', 'o', 't', 'k', 'o', 't']
```

```
l1 = itertools.chain.from_iterable([1, 2, 3])
print((list(l1)))
```

→

```
Traceback (most recent call last):
```

```
File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print((list(l1)))
```

```
TypeError: 'int' object is not iterable
```

▼ `compress(data, selectors)` - фильтрует элементы *data*, возвращая только те, которые содержат соответствующий элемент в селекторах (*selectors*), стоящих в True. Прекращает выполнение, когда либо данные, либо селекторы закончились. (finite iterator)

```
import itertools

selectors1 = [True, False, True, False]
l1 = itertools.compress([1, 2, 3, 4], selectors1)
print(list(l1))
selectors2 = [False, False]
l2 = itertools.compress([1, 0], selectors2)
print(list(l2))
selectors3 = [True, True]
l3 = itertools.compress([0, 0, 0], selectors3)
print(list(l3))
```

→

[1, 3]

[]

[0, 0]

▼ `dropwhile(predicate, iterable)` - выбрасывает элементы из итерируемого объекта до тех пор, пока предикат (*predicate*) имеет значение `True`, а затем возвращает каждый элемент. Итератор не вернет выходных данных, пока предикат не получит значение `False`.(finite iterator)

```
import itertools

greater_three = itertools.dropwhile(lambda item: item<3, [1, 2, 3, 4])
print(list(greater_three))
```

→

[3, 4]

```
import itertools

return_if_first_odd = itertools.takewhile(lambda item: item%2!=0, [1, 3, 5,
4, 7])
print(list(return_if_first_odd))

return_if_first_even = itertools.takewhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are True to the predicate until it meets False, then iteration stops and function returns the iterable with the f
```

```

irst results

return_if_first_odd = itertools.dropwhile(lambda item: item%2!=0, [1, 3, 5,
4, 7, 6, 8, 9])
print(list(return_if_first_odd))

return_if_first_even = itertools.dropwhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are False to the predicate until
meets True, then iteration stops and function returns the iterable with the
first True result and the rest after it

```

→

[1, 3, 5]
[2, 2, 4, 8]
[4, 7, 6, 8, 9]
[1, 2, 4]

- ▼ `takewhile(predicate, iterable)` - возвращает элементы из итерируемого объекта до тех пор, пока предикат имеет значение True (finite iterator)

```

import itertools

greater_three = itertools.takewhile(lambda item: item<3, [1, 2, 3, 4])
print(list(greater_three))

```

→

[1, 2]

```

import itertools

return_if_first_odd = itertools.takewhile(lambda item: item%2!=0, [1, 3, 5,
4, 7])
print(list(return_if_first_odd))

return_if_first_even = itertools.takewhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are True to the predicate until
meets False, then iteration stops and function returns the iterable with the
first results

```

```

return_if_first_odd = itertools.dropwhile(lambda item: item%2!=0, [1, 3, 5,
4, 7, 6, 8, 9])
print(list(return_if_first_odd))

return_if_first_even = itertools.dropwhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are False to the predicate until
meets True, then iteration stops and function returns the iterable with the
first True result and the rest after it

```

→

```

[1, 3, 5]
[2, 2, 4, 8]
[4, 7, 6, 8, 9]
[1, 2, 4]

```

- ▼ `filterfalse()` - фильтрует элементы итерируемого объекта, возвращая только те, для которых предикат имеет значение `False`. Если предикат равен `None`, он возвращает элементы, которые стоят в значении `False`. (finite iterator)

```

import itertools

filter_odd = itertools.filterfalse(lambda item: item%2==0, [1, 2, 3, 5, 4, 7,
6, 8])
print(list(filter_odd))

filter_none = itertools.filterfalse(None, [0, 1, 2, 3, 4])
print(list(filter_none))

```

→

```

[1, 3, 5, 7]
[0]

```

- ▼ `zip_longest(iterables, fillvalue=None)` - агрегирует элементы из каждого итерируемого объекта. Если итераторы имеют неравномерную длину, то на место пропущенных значений ставится *fillvalue*. Итерация будет продолжаться до тех пор, пока не закончится самый длинный итерируемый объект. (finite iterator)

```

import itertools

zipped = itertools.zip_longest(["Gucci", "Chanel", "Prada"], ["in stock", "not in stock", "in stock", "not in stock", "in stock", "not in stock", "in stock", "not in stock"])
print(list(zipped))

zipped1 = itertools.zip_longest(["cat", "dog", "mouse", "hamster", "bunny"], ["fed", "fed"], fillvalue="mb need to feed")
print(list(zipped1))

```

→

[('Gucci', 'in stock'), ('Chanel', 'not in stock'), ('Prada', 'in stock'), (None, 'not in stock'), (None, 'in stock'), (None, 'not in stock'), (None, 'in stock'), (None, 'not in stock')]
[('cat', 'fed'), ('dog', 'fed'), ('mouse', 'mb need to feed'), ('hamster', 'mb need to feed'), ('bunny', 'mb need to feed')]

- ▼ `starmap()` - вычисляет функцию, получая аргументы из итерируемого объекта.
Используется вместо `map()`, когда параметры аргумента уже сгруппированы в кортежи в одном итерируемом объекте (данные были предварительно сжаты).
(finite iterator)

```

import itertools

a2=itertools.starmap(lambda x:x**2,[(1, ),(2, ),(3, )])
print (list(a2))

sq = itertools.starmap(pow, [(1, 2), (2, 2), (3, 2)])
print(list(sq))

```

→

[1, 4, 9]
[1, 4, 9]

- ▼ `islice()` - возвращает выбранные элементы из итерируемого объекта.
Default `start` - 0, `step` - 1, `stop` - до конца итерируемого объекта. Если его нет, итератор остановится на определенной позиции. Значения `start`, `stop` и `step` >0.
(finite iterator)

```
import itertools

s1 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], None)
print(list(s1))
s2 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 5)
print(list(s2))
s3 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 3, None, 2)
print(list(s3))
s3 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 2, 7, 2)
print(list(s3))
```

→

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5]
[4, 6, 8]
[3, 5, 7]
```

- ▼ `tee(iterable, n=2)` - возвращает n независимых итераторов из одного итерируемого объекта. (finite iterator)

```
import itertools

l1 = [1, 2, 3, 4, 5, 5, 6]
l2 = itertools.tee(l1, 2)
for i in l2:
    print(list(i))
```

→

```
[1, 2, 3, 4, 5, 5, 6]
[1, 2, 3, 4, 5, 5, 6]
```

- ▼ `groupby(iterable, key=None)` - возвращает последовательно ключи и группы из итерируемого объекта (finite iterator)

key – это функция, вычисляющая значение ключа для каждого элемента по умолчанию. Если ключ не указан или в значении `None`, то по умолчанию ключ является функцией идентификации, которая возвращает элемент без изменений.

```

import itertools

l1 = [("size", "XS"), ("size", "S"), ("size", "M"), ("quantity", 120), ("quantity", 134), ("size", "L")]
l2 = itertools.groupby(l1, key=lambda x:x[0])
for key, group in l2:
    result = {key: list(group)}
    print(result)
print()
l3 = [("gender", "female"), ("gender", "male"), ("age", 18), ("age", 21), ("age", 34)]
l4 = itertools.groupby(l3)
for key, group in l4:
    result = {key: list(group)}
    print(result)

```

→

```

{'size': [('size', 'XS'), ('size', 'S'), ('size', 'M')]}

{'quantity': [('quantity', 120), ('quantity', 134)]}

{'size': [('size', 'L')]}

{('gender', 'female'): [('gender', 'female')]}

{('gender', 'male'): [('gender', 'male')]}

{('age', 18): [('age', 18)]}

{('age', 21): [('age', 21)]}

{('age', 34): [('age', 34)]}

```

- ▼ `product(iterables, repeat)` - декартово произведение итерируемых объектов, подаваемых на вход.(combinatory iterator)

Декартово произведение - произведение множества X и множества Y – это множество, содержащее все упорядоченные пары (x, y) , в которых x принадлежит множеству X , а y принадлежит множеству Y .

Чтобы вычислить произведение итерируемого объекта умноженного самого на себя, нужно указать количество повторений с помощью опционального аргумента с ключевым словом `repeat`. Например, `product(A, repeat=4)` – тоже самое, что и `product(A, A, A, A)`.

```

import itertools

```

```

#Only one iterable is given
l1=itertools.product("ABCD")
print (list(l1))

#two iterables are given
l2=itertools.product("ABC", [1,2])
print (list(l2))

#one iterable and repeat is mentioned.
l3=itertools.product("xy", repeat=2)
print (list(l3))

l4=itertools.product("aa", repeat=2)
print (list(l4))

#More than two iterables is mentioned
l5=itertools.product([1,2], [3,4], [5,6])
print (list(l5))

```

→

```

[('A',), ('B',), ('C',), ('D',)]
[('A', 1), ('A', 2), ('B', 1), ('B', 2), ('C', 1), ('C', 2)]
[('x', 'x'), ('x', 'y'), ('y', 'x'), ('y', 'y')]
[('a', 'a'), ('a', 'a'), ('a', 'a'), ('a', 'a')]
[(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)]

```

- ▼ permutations() - возвращает последовательные **r** перестановок элементов в итерируемом объекте. (combinatory iterator)

Если **r** нет или **None**, то **r** = длине итерируемого объекта и генерирует все возможные полноценные перестановки. Кортежи перестановок выдаются в лексикографическом порядке в соответствии с порядком итерации входных данных. Т. о., если входные данные итерируемого объекта отсортированы, то комбинация кортежей будет выдаваться в отсортированном порядке.

Элементы рассматриваются как уникальные в зависимости от их позиции, а не от их значения. Т. о., если входные элементы уникальны, то в каждой перестановке не будет повторяющихся значений.

```

import itertools

l1=itertools.permutations("ABC")
print (list(l1))

l2=itertools.permutations([3,2,1])
print (list(l2))

#elements are treated as unique based on their position and not by their value.
l3=itertools.permutations([1,1])
print (list(l3))

l4=itertools.permutations(["ABC"])
print (list(l4))

#r value is mentioned as 2. It will return all different permutations in 2 values.
l5=itertools.permutations([1,2,3,4],2)
print (list(l5))

l4=itertools.product("aa", repeat=2)
print (list(l4))

#More than two iterables is mentioned
l5=itertools.product([1,2],[3,4],[5,6])
print (list(l5))

```

→

```

[('A', 'B', 'C'), ('A', 'C', 'B'), ('B', 'A', 'C'), ('B', 'C', 'A'), ('C', 'A', 'B'), ('C', 'B', 'A')]
[(3, 2, 1), (3, 1, 2), (2, 3, 1), (2, 1, 3), (1, 3, 2), (1, 2, 3)]
[(1, 1), (1, 1)]
[('ABC',)]
[(1, 2), (1, 3), (1, 4), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (3, 4), (4, 1), (4, 2), (4, 3)]
[('a', 'a'), ('a', 'a'), ('a', 'a'), ('a', 'a')]
[(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)]

```

▼ **combinations()** - возвращает подпоследовательности длины **r** из элементов итерируемого объекта, подаваемого на вход. (combinatory iterator)

Комбинация кортежей генерируется в лексикографическом порядке в соответствии с порядком элементов итерируемого объекта на входе. Таким образом, если входной итерируемый объект отсортирован, то комбинация кортежей будет генерироваться в отсортированном порядке.

Лексикографический порядок – способ упорядочивания слов в алфавитном порядке.

Элементы рассматриваются как уникальные в зависимости от их позиции, а не значения. Таким образом, если выходные элементы уникальны, то в каждой комбинации не будет повторяющихся значений.

```
import itertools

l5=itertools.combinations([1,2,3,4],2)
print (list(l5))
```

→

`[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]`

- ▼ `combinations_with_replacement()` - возвращает подпоследовательности длины `r` из элементов итерируемого объекта, подаваемого на вход, при этом отдельные элементы могут повторяться больше одного раза.

```
import itertools
l1=itertools.combinations("ABC",2)
print (list(l1))
l1=itertools.combinations_with_replacement("ABC",2)
print (list(l1))

l2=itertools.combinations([3,2,1],3)
print (list(l2))
l2=itertools.combinations_with_replacement([3,2,1],3)
print(list(l2))

#elements are treated as unique based on their position and not by their value.
l3=itertools.combinations([1,1],2)
print (list(l3))
l3=itertools.combinations_with_replacement([1,1],2)
print (list(l3))
```

```

#since list contains only one element, given r value is 2. So it returns empty list.
l4=itertools.combinations(["ABC"],2)
print (list(l4))
#In combinations_with_replacement,it allows repeated element.
l4=itertools.combinations_with_replacement(["ABC"],2)
print (list(l4))

```

→

```

[('A', 'B'), ('A', 'C'), ('B', 'C')]
[('A', 'A'), ('A', 'B'), ('A', 'C'), ('B', 'B'), ('B', 'C'), ('C', 'C')]
[(3, 2, 1)]
[(3, 3, 3), (3, 3, 2), (3, 3, 1), (3, 2, 2), (3, 2, 1), (3, 1, 1), (2, 2, 2), (2, 2, 1), (2, 1, 1),
(1, 1, 1)]
[(1, 1)]
[(1, 1), (1, 1), (1, 1)]
[]
[('ABC', 'ABC')]

```

```

#r value is not mentioned. It will raise TypeError
l5=itertools.combinations([1,2,3,4])
print (list(l5))#Output:TypeError: combinations() missing required argument
'r' (pos 2)
l5=itertools.combinations_with_replacement([1,2,3,4])
print (list(l5))#Output:TypeError: combinations_with_replacement() missing required argument 'r' (pos 2)

```

▼ dir()

- ▼ There is a built-in function to list all the function names (or variable names) in a module - the `dir()` function.

```

import platform

x = dir(platform)
print(x)

```

→

```
['_Processor', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES',  
'builtins', 'cached', 'copyright', 'doc', 'file', 'loader', 'name', 'package', 'spec', 'version',  
'_comparable_version', '_component_re', '_default_architecture', '_follow_symlinks',  
'_get_machine_win32', '_ironpython26_sys_version_parser',  
'_ironpython_sys_version_parser', '_java_getprop', '_libc_search', '_mac_ver_xml',  
'_node', '_norm_version', '_platform', '_platform_cache', '_pypy_sys_version_parser',  
'_sys_version', '_sys_version_cache', '_sys_version_parser', '_syscmd_file',  
'_syscmd_ver', '_uname_cache', '_unknown_as_blank', '_ver_output', '_ver_stages',  
'architecture', 'collections', 'functools', 'itertools', 'java_ver', 'libc_ver', 'mac_ver',  
'machine', 'node', 'os', 'platform', 'processor', 'python_branch', 'python_build',  
'python_compiler', 'python_implementation', 'python_revision', 'python_version',  
'python_version_tuple', 're', 'release', 'subprocess', 'sys', 'system', 'system_alias', 'uname',  
'uname_result', 'version', 'win32_edition', 'win32_is_iot', 'win32_ver']
```

▼ Import from a module →

You can choose to import only parts from a module, by using the `from` keyword (from sm import sm_person).

When importing using the `from` keyword, do not use the module name when referring to elements in the module. Example: `person1["age"]`, not `mymodule.person1["age"]`

▼ Custom modules

▼ Attributes

▼ `__name__`

When Python is running in interactive mode or as a script, its `__name__` is `_main_`. It is the name of scope in which top level is being executed. However, for imported module this attribute is set to name of the Python script. (excluding the extension .py).

▼ `__doc__`

This attribute returns the docstring of module. Just as in function, Documentation string (docstring) is a string literal in first line written in module code.

▼ `__file__`

This attribute will returns the name and path of the module file.

▼ `__dict__`

This attribute returns a dictionary object of all attributes, functions and other definitions and their respective values.

▼ Module search

How does Python interpreter find a module when a script requests it to be imported? Python follows following method to locate a module:

- Whether it is present in current working directory?
- If not found there, directories in PYTHONPATH environment variable are searched.
- Otherwise, it searches the installation dependent default directory.

▼ Reloading a module

Python loads any module only once even if import statement is issued repeatedly.

```
"docstring of messages module"
def welcome(name):
    print ("Hi {}. Welcome to Python Tutorial".format(name))
    return
def bye(name):
    print ("Good Bye {}. See you again".format(name))
welcome("world")
bye("world")
```

Let us import messages module from Python prompt. The calls to functions in the code also get executed. However, repeating import statement doesn't execute them again.

```
>>> import messages
Hi world. Welcome to Python Tutorial
Good Bye world. See you again
>>> import messages
>>>
```

Now suppose we need to modify the SayHello() function before executing it again. Updated function is :

```
"docstring of messages module"
def welcome(name):
    print ("Hi {}. Welcome to Python Tutorial".format(name))
    return
def bye(name):
```

```
    print ("Good Bye {}. See you again".format(name))
welcome("Guest")
bye("Guest")
```

This change will be effected only if current interpreter session is closed and re-launched. If such changes are to be done frequently, to close and restart Python is cumbersome. In order to call load the module without terminating interpreter session, use reload() function from imp module.

```
>>> import imp
>>> imp.reload(messages)
Hi Guest. Welcome to Python Tutorial
Good Bye Guest. See you again
<module 'messages' from 'C:\\python37\\messages.py'>
```

Decorators

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-decorators>

CGI in Python

Common Getaway Interface - стандарт для внешних шлюзовых программ для взаимодействия с информационными серверами (пр.: HTTP-серверы). Это набор соглашений, к-й должен соблюдаться web-серверами при выполнении ими различных web-приложений.

Переменные окружения

- ▶ QUERY_STRING – храниться значения
- ▶ REQUEST_METHOD – храниться метод передачи данных get или post
- ▶ CONTENT_TYPE – хранит тип передачи данных
Для get обычно используется "application/x-www-form-urlencoded"
Для post обычно используется "multipart/form-data"
- ▶ HTTP_HOST – хранит имя хоста с портом
- ▶ SERVER_NAME - хранит имя хоста
- ▶ HTTP_USER_AGENT – хранит сведения о клиенте "Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0"
- ▶ HTTP_REFERER – формируется автоматически браузером и хранит url страницы с которой пришёл запрос

▶ CONTENT_LENGTH – хранит строку, являющуюся десятичным представлением длины данных в байтах. Присутствует только в методе POST, в GET отсутствует

▶ HTTP_COOKIE – хранит все cookies в URL-кодировке

▶ HTTP_ACCEPT – хранит перечисления типов данных документа который принимает браузер например: "text/html, text/plain, image/gif, image/jpeg", но все новые браузеры передают "*" – это значит что принимают все типы.

Подробнее: <https://andreyex.ru/yazyk-programmirovaniya-python/uchebnik-po-python-3/python-3-programmirovaniye-v-python-cgi/>

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-cgi>

Django

Документация: <https://docs.djangoproject.com/en/4.0/ref/contrib/messages/>

▼ Методика MTV (или MVC)

Описывает общий дизайн БД ориентированных веб-приложений Django.

Django поощряет свободное связывание и строгое разделение частей приложения. Поэтому можно легко вносить изменения в одну конкретную часть приложения без ущерба для остальных частей. Так, мы используем тот же принцип разделения, как если бы отделяли бизнес-логику от логики отображения с помощью шаблонной системы.

Эти три вещи вместе — логика доступа к данным, бизнес-логика и логика отображения — составляют концепцию, которую называют шаблоном *Модель-Представление-Управление*

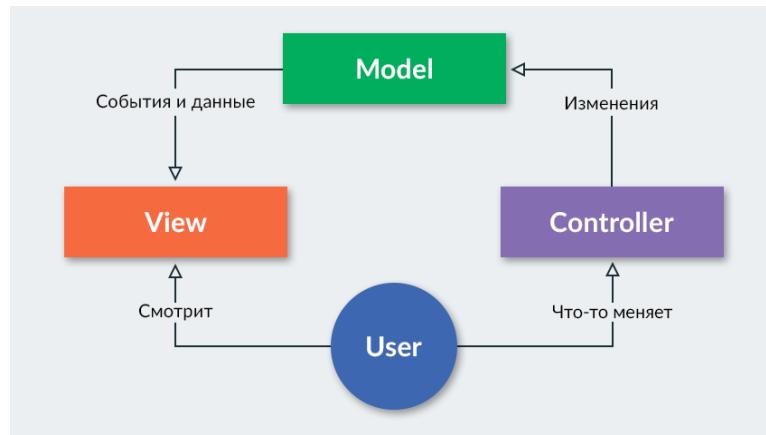
(*Model-View-Controller*, MVC) архитектуры программного обеспечения. В этой концепции термин «Модель» относится к логике доступа к данным; термин «Представление» относится к той части системы, которая определяет, что показать и как; а термин «Управление» относится к той части системы, которая определяет какое представление надо использовать, в зависимости от пользовательского ввода, по необходимости получая доступ к модели.

Django следует модели MVC достаточно близко, т.е., может быть назван MVC совместимой средой разработки. Вот примерно как M, V и C используются в Django:

- *M*, доступ к данным, обрабатывается слоем работы с базой данных, который описан в этой главе.
- *V*, эта часть, которая определяет какие данные получать и как их отображать, обрабатывается представлениями и шаблонами.
- *C*, эта часть, которая выбирает представление в зависимости от пользовательского ввода, обрабатывается самой средой разработки, следуя созданной вами схемой URL, и вызывает соответствующую функцию Python для указанного URL.

Так как «С» обрабатывается средой разработки и всё интересное в Django происходит в моделях, шаблонах и представлениях, на Django ссылаются как на *MTV-ориентированную среду разработки*. В MTV-подходе к разработке:

- *M* определено для «Модели» (Model), слоя доступа к данным. Этот слой знает всё о данных: как получить к ним доступ, как проверить их, как с ними работать и как данные связаны между собой.
- *T* определено для «Шаблона» (Template), слоя представления данных. Этот слой принимает решения относительно представления данных: как и что должно отображаться на странице или в другом типе документа.
- *V* определено для «Представления» (View), слоя бизнес-логики. Этот слой содержит логику, как получать доступ к моделям и применять соответствующий шаблон. Вы можете рассматривать его как мост между моделями и шаблонами.



REST Framework

CORS headers

▼ Протоколы прикладного уровня

- [9P](#)
- [BitTorrent](#)
- [BOOTP](#)
- [DNS](#)
- [FTP](#)
- [HTTP](#)

- NFS
- POP, POP3
- IMAP
- RTP
- SMTP
- SNMP
- SPDY
- Telnet
- SSH
- X.400
- X.500
- RDP
- Matrix
- SFTP

Компьютерные сети:

- по типу коммутации: комм. каналов (по каналу, пр.: тел. сети) и комм. пакетов (каждый пакет по своему пути, пр.: комп. сети)
- по технологии передача: широковещательные сети (сразу всем устр.) и “точка-точка”(от одного другому, мб посредством третьего и т.д.)
- протяженность
(персональная → локальная → муниципальная → глобальная → объединение сетей)

Топология КС - способ объединения комп. в сеть (еще наз. “граф”)

Вершины графа - узлы сети (комп. и сет. устр), ребра - связи между узлами (физ. и инф.)

- полносвязная Т - каждый соед. с каждым
- ячеистая Т - как полносвязная, но без некоторых соед.
- звезда - все подсоединенены к центральному устр-ву (коммутатор, ви-фи, маршрутизатор и т.д.), через к-е и осущ. передача данных

- кольцо - передача по кругу через соседей
- дерево - комп. и ЦУ обр-т дерево (ЦУ → др. ЦУ → комп.)
- общая шина - все комп. подкл к общ. сети передачи (пр. медный кабель), как к каналу
- смешанная - (на практике) - пр.: ЦУ в кольце, к ним подкл-ы комп-ы по звезде, а ост-е - по дереву через доп. ЦУ.

Физич. Т - соед. устр-в в сети. Логич. Т - правила расп-я сигналов в сети. Могут различаться: Ethernet - физ-ки звезда, лог-ки шина. Коммутир. Ethernet - ф-ки звезда, л-ки полносвяз. WiFi - физ. - нет, лог. - шина.

Стандарты сетей:

- Эталонная модель взаимодействия открытых сетей (принята OSI, м-н орг. по стандартизации)
- Технологии передачи данных (институт инж. по эл-ке и электротехнике, IEEE)
- Протоколы интернет (прин. Совер по арх. интернета, IAB)
- стандарты Web (консорциум W3C)

Для решения сложностей в организации масштабируемой КС был создан метод декомпозиции задач “уровни”. У кажд. ур. своя задача (или набор связ. задач).

Сервис - опис-т фу-ии ур-я.	Интерфейс - набор примитив. опер., к-е ниж. ур. предост. верх.	Протокол - прав. и соглаш. для связи ур. N 1 комп. с ур. N 2 комп.
--------------------------------	--	---

Арх. сети - набор ур. и протоколов (интерфейсы не входят). Стек протоколов собран иерархически.

Модель OSI

Хорошая теор. детализация, но на практ не прим-ся. Open Systems Interconnection, Модель взаимодействия открытых систем. Октр. сист. - в соотв. с открытыми спецификациями.

7. Ур. приложения (сообщение) - он же прикладной - включает все сервисы, к-ми может пользоваться пользователь (гипертекстовые web-страницы, соц. сети, видео и аудио связь, эл. почта, доступ к разделяемым файлам и т.д.).

Здесь располагаются сетевые службы, позволяющие пользователю взаимодействовать с машиной: Telnet, LPD, TFTP, NFS, DNS, DHCP, SNMP, X Window.

- Протокол HTTP (Hyper-Text Transfer Protocol) - протокол передачи гипертекста, основа www.

URL (Uniform Resource Locator) - уникальное положение ресурса. HTTP работает в режиме запрос-ответ.

Постоянное TCP соединение служит для того, чтобы по протоколу HTTP передавалось не по 1 файлу за соединение, а сразу все. (keep-alive или persistent connection).

В HTTP 1.0 нужно добавить строчку Connection: keep-alive, а в HTTP 1.1 все соединения по умолчанию постоянные (чтобы разорвать, нужно добавить заголовок Connection: close)

Минусы: клиент открыл соединение и не использует его - ресурсы недоступны другим, плохо для высоконагруженных серверов. Решение: автоотключение через таймаут 5-15 сек., или еще конвейреная обработка (pipelining) (несколько запросов → несколько ответов, недостаток - сохранение порядка, решена в HTTP2, где есть нумерация запросов), или еще вариант - несколько HTTP соединений (каждое м.б. постоянным или использовать конвейерную обработку). Последний вариант сейчас чаще всего используется.

Кэширование сокращает время загрузки страницы, но требует место на лок. диске комп-а. Может кэшировать отдельные ресурсы, к-е редко меняются.

Заголовок Expires помогает понять, можно ли взять страницу из кэша или ее нужно обновить. Если в браузере такого нет, используются эвристики (Last-Modified). Другой способ (нивелирующий ошибки предыдущих) - GET Conditional (при наличии Last-Modified): были ли изменения? В совр. версиях HTTP также используется ETag (entity tag) в запросах GET с условием.

Заголовок Cache-Control служит для управления кэшем.

Альтернатива хранению на лок. диске комп-а - прокси-сервер с разделяемым кэшем.

Есть еще обратный прокси, к-й устанавливается не со стороны клиента, а со стороны вэб-серверов.

Порядок следования HTTP сообщение:

- HTTP-метод
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (<http://>), домена (здесь developer.mozilla.org), или TCP порта (здесь 80)
- Версию HTTP-протокола

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера
- Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс

Методы HTTP

GET – запрос Web-страницы
POST – передача данных на Web-сервер
HEAD – запрос заголовка страницы
PUT – помещение страницы на Web-сервер
DELETE – удаление страницы с Web-сервера
TRACE – трассировка страницы
OPTIONS – запрос поддерживаемых методов HTTP для ресурса
CONNECT – подключение к Web-серверу через прокси

Статусы HTTP

1XX – информация
2XX – успешное выполнение (200 OK)
3XX – перенаправление (301 – постоянное перемещение, 307 – временное перенаправление)
4XX – Ошибка на стороне клиента (403 – доступ запрещен, 404 – страница не найдена)
5XX – Ошибка сервера (500 – внутренняя ошибка сервера)

Структура пакета HTTP

→ Запрос/статус ответа

- GET /courses/networks
- 200 OK

Заголовки (не обязательно)

- Host: www.asozykin.ru (обязательно в HTTP 1.1)
- Content-Type: text/html; charset=UTF-8
- Content-Length: 5161

Тело сообщения (не обязательно)

- Страница HTML
- Параметры, введенные пользователем

Пример запроса HTTP

Подключение по TCP к серверу www.asozykin.ru,
порт 80

```
-----  
GET /courses/networks HTTP/1.1  
Host: www.asozykin.ru
```

Пример ответа HTTP

```
HTTP/1.1 200 OK  
Server: nginx  
Content-Type: text/html; charset=UTF-8  
Content-Length: 5161
```

```
<html lang="ru-RU">  
<head>  
...  
</html>
```

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь `developer.mozilla.org`), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь `developer.mozilla.org`), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь developer.mozilla.org), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

- Протокол SMTP (Simple Mail Transfer Protocol) - простой протокол передачи почты (посл. версия - ESMTP, extended). Схему использования протокола при передаче почты - см. ниже.

Теоретически SMTP может использовать любой транспортный протокол (TCP, UDP, др.). Порты: 25 для передачи почты между серверами, 587 для приема почты от клиентов. На практике: протокол TCP, порт 25.

Протокол SMTP входит в конверт письма, а заголовки и содержимое письма включают RFC 2822.

Протокол работает в формате текста в режиме запрос-ответ.

Минусы протокола: незащищенность данных, спам.

Команды SMTP

Команда	Назначение	Пример
HELO	Установка соединения	HELO example.com
MAIL	Адрес отправителя	MAIL FROM: sender@example.com
RCPT	Адрес получателя	RCPT TO: recipient@mail.ru
DATA	Передача письма	DATA
QUIT	Выход	QUIT

Ответы SMTP

Код	Назначение	Пример
220	Подключение к серверу успешно	220 smtp.example.com ESMTP Postfix
250	Успешное выполнение предыдущей команды	250 Hello example.com 250 Ok
354	Начало передачи письма	354 End data with <CR><LF>.<CR><LF>
502	Команда не реализована	502 5.5.2 Error: command not recognized
503	Неправильная последовательность команд	503 5.5.1 Error: need MAIL command
221	Закрытие соединения	221 2.0.0 So long, and thanks for all the fish

Заголовки письма

Заголовок	Назначение
From:	Отправитель (имя и адрес)
To:	Получатель
CC:	Получатель копии письма
BCC:	Получатель копии, адрес которого не должен быть показан
Reply-To:	Адрес для ответа
Subject:	Тема письма
Date:	Дата отправки письма

- Протокол IMAP (Internet Message Access Protocol) - протокол доступа к эл. почте. Письма хранятся на почтовом сервере. Клиенты подключаются к серверу и загружают письма только после запроса пользователя. Сервер может выполнять сложные операции с письмами.

Преимущества: одновременно могут работать несколько клиентов, все клиенты видят одно и то же состояние почтового ящика.

Недостатки: более сложный протокол, ограниченное место для почтового ящика на сервере.

IMAP использует TCP, порт 143.

IMAP позволяет использовать несколько почтовых ящиков и папок (хранятся на сервере, можно перемещать и образовывать иерархию). Также есть флаги.

Состояния сеанса: клиент не аутентифицирован → клиент аутентифицирован → папка выбрана → выход.

Работает в текстовом режиме, взаимодействие запрос-ответ. Позволяет выполнять несколько команд одновременно (есть теги команд).

Статусы: OK, NO (ошибка), BAD (неправильная команда).

- Протокол POP3 (Post Office Protocol) - протокол почтового отделения. Работает по принципу загрузить и удалить. Ящик на сервере - временное хранилище, сообщения переписываются на почтовый клиент, после чего удаляются с сервера.

Плюсы: простота, письма доступны при отсутствии подключения к сети. Минусы: только один клиент, единое хранилище писем (нет папок, фильтров и т.д.).

Состояния сеанса: авторизация, транзакция, обновление.

Протокол работает в текстовом режиме, взаимодействие запрос-ответ.

Ответы протокола: + OK или - ERR.

Команды POP3

Команда	Назначение	Пример
USER	Указать имя пользователя	USER asozykin
PASS	Указать пароль	PASS 1234qwer
STAT	Количество писем на сервере	STAT
LIST	Передача информации о сообщениях	LIST 2
RETR	Передать сообщение на клиент	RETR 1
TOP	Передать на клиент заголовок сообщения	TOP 2 10
DELE	Пометить сообщение на удаление	DELE 1
QUIT	Закрытие транзакции, удаление сообщений и отключение	QUIT

- **Протокол DNS** (Domain Name System) - система доменных имен. Позволяет использовать вместо неудобных IP понятные для пользователя доменные имена. Также позволяет менять сетевую инфраструктуру (при переходе на другой IP домен менять не надо). Один домен может обслуживать несколько серверов. Узнать IP по домену можно с помощью утилиты nslookup. Для Linux это утилиты host и dig.

Особенности DNS:

- распределенная (децентрализованная) система (нет единого сервера, на котором описываются имена хостов)
- делегирование ответственности (пространство имен разделено на отдельные части - домены, за каждый домен твердит отдельная организация)
- надежность (дублирование серверов DNS)

Структура корневого домена: www(имя компьютера).wiki(домен второго уровня).com(домен верхнего уровня).(корневой домен)

Важная особенность - делегирование ответственности за хранение данных между доменами разных уровней.

Распределением имен занимаются регистраторы.

Режимы работы DNS:

1) Итеративный:

- если сервер отвечает за данную доменную зону - он возвращает ответ
- если нет - возвращает адрес DNS-сервера, у которого есть более точная информация

2) Рекурсивный:

- сервер сам выполняет запросы к другим серверам DNS, чтобы найти нужный адрес
Сервер разрешения имен DNS предоставляет провайдером/организацией, комп.
получает адрес локального DNS по DHCP.

Как только адрес найден, DNS сервер записывает его в кэш.

Типы ответа DNS:

- 1) авторитетный: ответ от сервера, обслуживающего доменную зону; получен из файлов на диске сервера
- 2) неавторитетный: ответ от сервера, не обслуживающего доменную зону; получен из кэша, данные могли устареть.

Работает по модели клиент-сервер. Взаимодействие идет в режиме запрос-ответ. DNS использует протокол UDP, номер порта 53.

Что еще может DNS:

- определять для доменного имени адреса IPv4 и IPv6
- задавать несколько доменных имен для одного IP-адреса
- находить адрес почтового сервера для домена
- определять IP-адрес и порт некоторых сетевых сервисов
- задавать адрес DNS-серверов для доменной зоны
- определять по IP-адресу доменное имя

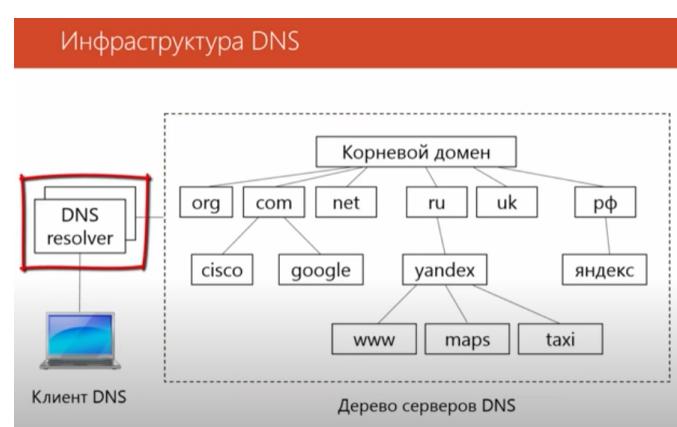
Запись MX нужна для отправки почты на почтовый сервер.

Чтобы присвоить несколько имен: псевдоним или несколько A записей.

Для некоторых сервисов можно задавать не только IP-адреса, но и номера портов. Для этого используется SRV запись.

Запись типа NS (name server) указывает адреса DNS серверов, отвечающих за зону.

Записи типа PTR используются реверсивной зоной для отпределения имени по IP.



Формат пакета DNS



- Протокол FTP (File Transfer Protocol) - протокол передачи файлов.

Работает по модели клиент-сервер. На сервере есть структура файлов. Клиент по протоколу подключается к серверу и работает с файлами.

Как и HTTP использует URL (`ftp://ftp-server.ru/path/file.format`). Использует два соединения: управляющее и для передачи данных.

Использует TCP, порт 21. Соединение данных: в активном режиме - порт сервера 20, в пассивном - порты больше 1024 (исп. если между клиентом и сервером есть интерфейс, напр. экран).

Есть аутентификация и режим анонимса.

Недостатки: проблемы с NAT (решает пассивный режим), низкая безопасность.

Замена: протоколы на основе SSH (SFTP, SCP).

Протокол FTP

Команда	Назначение
USER	Указать имя пользователя
PASS	Указать пароль
LIST	Просмотр содержимого каталога
CWD	Смена текущего каталога
RETR	Передать файл с сервера на клиент
STOR	Передать файл с клиента на сервер
TYPE	Установить режим передачи
DELE	Удалить файл
MKD	Создать каталог
RMD	Удалить каталог
PASV	Использовать пассивный режим
QUIT	Выход и разрыв соединения

6. Ур. представления (сообщение) - обеспечивает согласование синтаксиса и семантики передаваемых данных (форматы представления символов, форматы чисел). Шифрование и дешифрование. Пр.: Transport Layer Security (TLS), Secure Sockets Layer (SSL). Тут происходит кодирование и сжатие (пр.: JPEG, GIF).

Для защиты передаваемых по сети данных часто используется шифрование:

- Secure Socket Layer (SSL)
- Transport Layer Security (TLS)

Протоколы, которые используют SSL/TSL:

- HTTPS, порт 443
- IMAPS, порт 993
- SMTPS, порт 465
- FTPS

5. Сеансовый (сообщение) - позволяет устанавливать сеансы связи. Управляет диалогом (очередность передачи сообщений). Управляет маркерами (предотвращение одновременного выполнения критичной операции). Синхронизация (метки в сообщениях для возобновления передачи в случае сбоя).

Сеанс (сессия) - набор связанных между собой сетевых взаимодействий, направленных на решение одной задачи.

Загрузка Web-страницы:

- загрузка текста (.html)
- загрузка стилевого файла (.css)
- загрузка изображений

Подходы к загрузке Web-страницы:

- для каждого элемента создается отдельное соединение (HTTP 1.0)
- загрузка всех элементов через одно соединение TCP (HTTP keep-alive)

Пример взаимодействия на сеансовом уровне: аудио и видео конференция, т.к. на этом уровне устанавливается, каким кодаком будет кодироваться сигнал (кодаки должны совпадать с обеих сторон).

4. **Транспортный** (сегмент, дейтаграмма) - обеспечивает передачу данных между процессами на хостах. Надежность выше сети, гарант. порядок сообщений. TCP (все в точности) и UDP (может что-то потеряться). Сквозной уровень, т.к. дотавляет сообщения от источника адресату, в то время как предыдущие уровни исп. принцип звеньев (т.ж. тр. ур. называют сетенезависимым).

У хостов есть все 7 уровней, а вот у сетевого оборудования - первые 3. Транспортный уровень может связывать хосты, минуя сетевое оборудование - сквозное соединение. Важно указать адресацию, чтобы понимать, для какого процесса предназначен пакет. Для адресации используются порты. Это число от 0 до 65535. Каждое сетевое приложение на хосте имеет свой порт. Номера портов у приложений не повторяются. Формат записи: IP:порт.

Хорошо известные порты:

- 80 - HTTP (Web)
- 25 - SMTP (email)
- 53 - DNS
- 67, 68 - DHCP
- использовать может только root/ админ

Зарегистрированные порты: 1025-49151 (рег. в IANA). Динамические порты автоматически назначаются ОС клиенту сетевым приложениям.

Так, например, если мы (клиент) откроем браузер 1 и сделаем запрос на web-сервер (daemon), то сервер увидит наш IP и порт, и отправит ответ на порт браузера 1. Если откроем браузер 2 и сделаем запрос, ответ придет уже на порт браузера 2, который автоматически присвоила ему наша ОС при открытии.

Надежность уровня:

- гарантированная доставка данных:
 - подтверждение получения
 - данные отправляются снова, если не было подтверждения доставки
- гарантированный порядок следования сообщений - нумерация сообщений

Для взаимодействия с транспортным уровнем используется интерфейс сокетов:

Интерфейс транспортного уровня TCP/IP



UDP (User Datagram Protocol) - протокол действа грам пользователя. (дейтаграмма - сообщение UDP).

Особенности:

- нет соединения, за счет этого быстрее TCP
- нет гарантии доставки данных
- нет гарантии сохранения порядка сообщений

Зачем нужен? На транспортном уровне необходимо указать порты отправителя и получателя, что и делает протокол.

По надежности: в совр. сетях ошибки редки, да и обработать их могут приложения

Область применения: клиент-сервер и короткие запросы-ответы. Пример: DNS (клиент-DNS - сервер-DNS).

TCP (Transmission Control Protocol) - протокол управления передачей.

Обеспечивает надежную передачу потока байт (reliable byte stream).

Гарантии: доставка данных и сохранение порядка следования сообщений.

В протоколе TCP поток данных делится на отдельные сегменты. Они отправляются отдельно к получателю, который в свою очередь собирает их обратно в поток байт.

Чтобы обеспечить гарантию передачи данных TCP использует подтверждение получения.

Также TCP использует метод скользящего окна, подтверждая не каждый сегмент, а совокупность.

Для гарантии сохранения порядка отправления, TCP нумерует байты (сегментами по 1024

байт с 0 байт). Если произошла ошибка при отправке подтверждения о получении, получатель видит, что у него уже есть этот фрагмент и повторно отправляет подтверждение.

Перед передачей данных TCP необходимо установить соединение (это замедляет процесс).

Задачи соединения:

- убедиться, что отправитель и получатель хотят передавать друг другу данные
- договориться о нумерации потока байт
- договориться о параметрах соединения (макс. размер сегмента и т.д.)

После завершения передачи данных соединение разрывается.

Варианты подтверждения доставки:

- остановка и ожидание (WiFi, канальный ур.): отправка данных → ожидание подтверждения получения → подтверждение получения → продолжение отправки данных и т.д. (медленно, локальный)
- скользящее окно (TCP, транспортный уровень): отправка нескольких порций данных без ожидания подтверждения получения → кумулятивное подтверждение (получил последнюю порцию данных и все предыдущие) (быстро, для больших сетей)
Размер окна - кол-во байт данных, к-е могут быть переданы без получения подтверждения.

Еще есть выборочное подтверждение (Selective Acknowledgement, SACK) - подтверждение диапазонов принятых байт, эффективно при большом размере окна, доп. поле заголовка TCP (параметр).

Установление соединения:

- запрос на соединение (SYN) + порядковый номер передаваемого байта
- подтверждение соединения + информ. о предыдущих принятых байтах + ACK с номером ожидаемого байта
- подтверждение получения подтверждения о соединении с номером предыдущего полученного байта + ACK с номером следующего к передаче байта
- соединение установлено

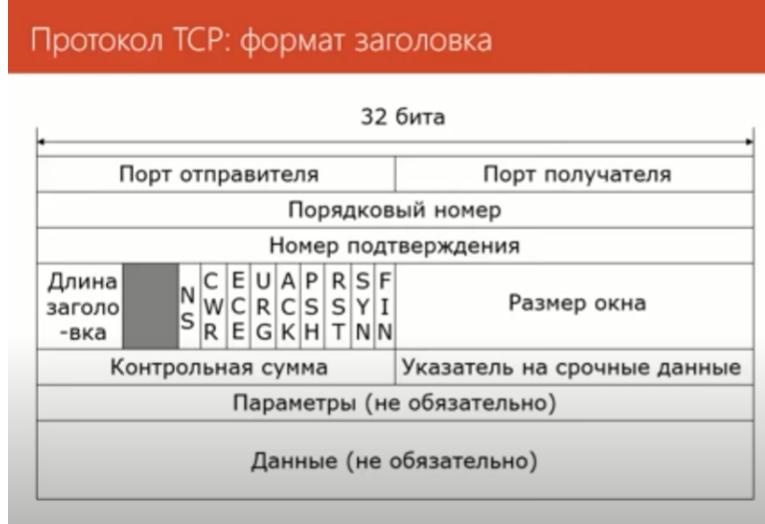
Разрыв:

- одновременное (обе стороны разорвали соединение)

- одностороннее (одна сторона прекращает передавать данные, но может принимать)

Есть варианты разрыва соединения: одностороннее (FIN) и из-за критической ситуации (RST, в обе стороны)

Порядок: FIN1 → ACK2 → (K2 закончил передачу данных) FIN2 → ACK1.



Управление потоком - справляется с проблемой “затопления”. Для этого есть поле “размер окна”.

Борьба с перегрузкой: окно перегрузки, которое рассчитывается в зависимости от нагрузки на сеть. В TCP есть AIMD (Additive increase/multiplicative decrease) - аддитивное увеличение/мультипликативное уменьшение - определяет размер окна перегрузки в зависимости от того, есть или нет перегрузки, опираясь на макс. размер сегмента. Сигнал о перегрузке - потеря сегмента или задержка сегмента или сигнал от маршрутизатора. Альтернатива - медленный старт: малый первоначальный размер окна перегрузки, при каждом подтверждении отправляется 1 сегмента, происходит экспоненциальный рост размера окна, а после сигнала о перегрузке все начинается с начала.

Интерфейс сокетов - для работы программиста с ПО на транспортном уровне (файлы).

Операции сокетов Беркли

Операция	Назначение
Socket	Создать новый сокет
Bind	Связать сокет с IP-адресом и портом
Listen	Объявить о желании принимать соединения
Accept	Принять запрос на установку соединения
Connect	Установить соединение
Send	Отправить данные по сети
Receive	Получить данные из сети
Close	Закрыть соединение

Работает по клиент-серверной модели. Сервер - программа, которая работает (слушает) на известном IP-адресе и порте и пассивно ждет запросов на соединение. Клиент - приложение, которое активно устанавливает соединение с сервером на заданом IP и порте.

NAT (Network Address Translation) - трансляция сетевых адресов (решение проблемы нехватки адресов IPv4). Это технология преобразования IP-адресов внутренней (частной) сети в IP-адреса внешней сети (Интернет).

Типы:

- статический (сколько комп=ов, столько и адресов, подх. для подключения к др. корп. сети)
- динамический (есть несколько IP для внеш. сети, которые поочередно используются комп-ами)
- один ко многим (masquerading) (один адрес на всех)

Межсетевой экран - отделяет сеть от других сетей (он же брандмауэр, firewall).

Перехватывает и проверяет пакеты.

Как обезопасить:

- флаг ACK - злоумышленник не сможет установить соединение (транспортный уровень)
- разрешить получение пакетов только от того, с кем установлено соединение (транспортный уровень)
- фильтрация на портах коммутатора по MAC-адресам (канальный уровень)

- прокси-сервер (прикладной уровень)
- фильтр содержимого (прикладной уровень)
- система обнаружения вторжений (IDS)
- система предотвращения вторжений (IPS)

Может замедлять и затруднять (а то и сделать невозможной) работу компьютера.

3. **Сетевой** (пакет)- объединяет сети, построенные на осн. разных технологий.

Обеспечивает: создание составной сети, согласование различий в сетях; адресацию (сетевые и глобальные адреса); опр. маршрута пересылки пакетов в сост. сети (маршрутизация). Оборудование: маршрутизатор. Использует IP адреса.

Объединяет сети разл. технологий (Ethernet, WiFi, 3/4/5G, MPLS). Этот уровень позволяет более удобно организовать масштабную сеть, чем с вифи и езернет.

Масштабируемость на сетевом ур.:

- агрегация адресов: работа с блоками адресов (блок адресов - сеть)
- запрет пересылки “мусорных” пакетов: т.е. тех, которые непонятно, куда отправлять
- возможность наличия неск. путей в сети: допускается неск. активных путей задача выбора лучшего пути - маршрутизация

Итого, сетевой уровень решает задачи: объединения сетей, маршрутизации, обеспечения качественного обслуживания.

Протоколы:

- **IP** - протокол передачи данных.

Передача данных: без гарантии доставки, без сохранения порядка следования сообщений. Протокол IP использует передачу данных без установки соединения.

Задачи: объединение сетей, маршрутизация, качество обслуживания.

Маршрутизация - поиск маршрута доставки пакета между сетями через транзитные узлы (маршрутизаторы).

Фрагментация - разделение пакета на несколько частей (фрагментов) для передачи по сети с маленьким MTU (max. transmission unit).

Формат заголовка IP-пакета

4 бита Номер версии	4 бита Длина заголовка	8 бит Тип сервиса	16 бит Общая длина			
		16 бит Идентификатор пакета	3 бита Флаги	13 бит Смещение фрагмента		
	8 бит Время жизни	8 бит Тип протокола	16 бит Контрольная сумма			
32 бита IP-адрес отправителя						
32 бита IP-адрес получателя						
Опции и выравнивание (не обязательно)						

- ICMP (управляющий протокол, доп.) - Internet Control Message Protocol - протокол межсетевых управляющих сообщений. Служит для управления сетью. Позволяет получать сообщения об ошибках сети (получатель недоступен, закончилось TTL - время жизни пакета, запрещено фрагментировать пакет). Служит для тестирования работы сети: ping (проверка доступности получателя) и traceroute (определение маршрута к получателю). Т.о. он нивелирует недостатки протокола IP (возможная потеря данных без дальнейшего восстановления передачи и без оповещения об ошибке). Есть утилита PING, к-я помогает обпределить наличие ошибок сети.

Типы ICMP- сообщений

Тип	Назначение сообщения
0	Эхо-ответ
3	Узел назначения недостижим
5	Перенаправления маршрута
8	Эхо-запрос
9	Сообщение о маршрутизаторе
10	Запрос сообщения о маршрутизаторе
11	Истечение времени жизни пакета
12	Проблемы с параметрами
13	Запрос отметки времени
14	Ответ отметки времени

Коды ICMP- сообщений (для типа 3)

Код	Причина
0	Сеть недостижима
1	Узел недостижим
2	Протокол недостижим
3	Порт недостижим
4	Ошибка фрагментации
5	Ошибка в маршруте источника
6	Сеть назначения неизвестна
7	Узел назначения неизвестен
8	Узел-источник изолирован
9	Административный запрет

- **ARP** (управляющий протокол, доп.) - Address Resolution Protocol - чтобы по IP определить протокол канального уровня. Протокол разрешения адресов. Связывает сетевой и канальный уровни. Позволяет определить по IP-адресу компьютера его MAC адрес.

Таблица соответствия (пр.: Linux “\$ cat /etc/ethers”): IP-MAC. В крупных сетях - ARP. Работает в режиме “широковещательный запрос-ответ компьютера, узнавшего свой IP”. Кеширование в ARP-таблицу (команда arp - a) на стороне отправителя.

Т.к. ниже сетевого уровня, пакеты ARP не проходят через маршрутизатор.

Зачем узнавать MAC-адрес? Сама технология (пр. Ethernet) не знает о том, что такое IP-адрес и для отправки информации ей нужен MAC.

- **DHCP** (управляющий протокол, доп.) - Dynamic Host Configuration Protocol - чтобы авто назначать IP комп-ам в составной сети. Протокол динамической конфигурации хостов. Модель “клиент-сервер”. Discover → Offer → Request → Ack (acknowledgement).

DHCP сервер должен быть установлен в той подсети, где находится клиент, т.к. пакеты DHCP не проходят через маршрутизатор (нет широковещания). Решение: DHCP-Relay сервер.

Устройство: маршрутизатор. Протоколы маршрутизации: BGP, OSPF, RIP, EIGRP.

2. **Канальный** (кадр, фрейм) - выделяет во входящем потоке бит отдельные сообщения, обнаруживает и корректирует ошибки. В широковещат. сетях еще обесп. физическую адресацию (указание, какому комп-у это нужно передать), а также управление доступом к среде передачи данных (в один момент вр. передает один комп.). Оборудование: коммутатор, точка доступа. Использует MAC адреса.

Передача данных: с сетевого уровня устройства 1 поступает пакет на канальный уровень устройства 1, канальный уровень присваивает пакету заголовок канального уровня и концевик, и через физический уровень передает этот кадр на канальный уровень устройства 2, который считывает заголовок и концевик, извлекает пакет сетевого уровня и передает своему сетевому уровню устройству 2.

Как канальный уровень определяет кадр (методы): указатель кол-ва бит (не на практике), вставка бит/байт (сейчас: протоколы HDLC и PPP: 1). 0(1x6)0 начало и конец кадра; 2). после 1ч5 в данных добавляется 0), устр-ва физ. уровня (преамбула Ethernet классического и избыточный код Fast Ethernet нового).

Подуровни: п/у управления логическими каналами (LLC) (передача данных, мультиплексирование, управление потоком, общий для разных технологий), п/у управления доступом к среде (MAC) (совместное использование разделяемой среды, адресация, специфичный для различных технологий, не я-ся обязательным).

Ethernet:

- на канальном и физическом ур.
- протокол STP (Spanning Tree Protocol): авто отключение дублир. соединений, связующее дерево, обесп. защиту от сбоев (широковещательного штурма при образовании кольца), надежность соед. между коммутаторами (новый - RSTP, rapid)

WiFi:

- на физ. и канал. ур, подуровни канал. ур.: п/у управл. логич. каналом (LLC) и п/у управления доступом к среде (MAC).
- типы кадров: кадры данных (передача данных), к. контроля (служебные и RTS, CTS, ACK), к. управления (реа-ия сервисов вифи, пр.: точка доступа)

1. **Физический** (бит)- передает биты единым потоком в виде сигналов по физ. каналу связи, не анализируя. Оборудование: концентратор. Здесь: Ethernet, WiFi, Bluetooth, инфракрасный порт. Сетевые устройства: концентраторы и репиторы.

Как данные переходят с канального на сетевой уровень?

- внутри одной сети:
 - есть K1 с IP1 и MAC1, K2 с IP2 и MAC2. В таблицу отправления вписываются данные отправителя, из данных получателя обычно известен только IP. По ARP узнаем его MAC. От получателя получаем ответ - такую же таблицу, только в зеркальном варианте.

- разные сети:
 - есть K1 с IP1 и MAC1, K2 с IP2 и MAC2. В таблицу отправления вписываются данные отправителя, из данных получателя обычно известен только IP. Проверяем, в одной ли сети: берем адрес сети отправителя и по ней проверяем по ней адрес получателя (используется маска подсети).
 - Т.к. в разных сетях, отправляем данные сначала на маршрутизатор, адрес которого знаем из таблицы маршрутизации. При помощи ARP узнаем MAC маршрутизатора. Маршрутизатор получает пакет и формирует свою таблицу для передачи: IP остаются, а MAC адреса меняются (отправитель - маршрутизатор с интерфейсом на стороне получателя, получатель - выявляем по ARP).
 - Получатель передает ответ - таблицу в которой IP-получатель - компьютер-отправитель, MAC-получатель - MAC сети, отправитель - данные отправителя. Получатель отправляет этот пакет на маршрутизатор.
 - Маршрутизатор снова меняет: отправитель имеет IP компьютера-получателя, а MAC - MAC сети со стороны отправителя; получатель - данные получателя пакета-подтверждения.

Итого: IP всегда остается, а MAC меняется.

Модель TCP/IP

Прим-ся на практик. (т.к. удобный стек протоколов)

4. Прикладной (сеансовый+представления+прикладной) - протоколы: HTTP, SMTP, DNS, FTP.
3. Транспортный - протоколы: TCP, UDP.
2. Интернет (сетевой) - протоколы: IP (доп.: ICMP, ARP, DHCP).
1. Сетевых интерфейсов (канальный+физический) - протоколы: Ethernet, WiFi, DSL.

Инкапсуляция - вкл-е сообщения вышестоящего ур. в сообщение нижестоящего. Т.е. по мере транспортировки данных с верхнего ур. до нижнего (от польз. интерфейса к машине). происх-т инкапсуляция, а обратно - декапсуляция (от машины к польз-ю). Сообщение = заголовок+данные+концевик.

IP адрес

Сетевой уровень.

Локальные адреса

- адреса в технологии канального уровня (пр.: MAC адрес в Ethernet, IMEI адрес в 4G)
- привязаны к конкретной технологии
- не могут быть использованы в гетерогенных сетях

Глобальные адреса

- адреса сетевого уровня (пр.: IP-адреса)
- не привязаны к технологии
- применяются при объединении сетей (Интернет)

Исп-ся для уникальной идентификации комп-ов в составной сети Интернет.

Версии протокола IP:

- IPv4: 4 байта
- IPv6: 16 байт

Сетевой уровень исп-т агрегацию адресов: масштабирование - работа не с отдельными адресами, а с подсетями. (поср-вом маршрутизаторов)

Подсеть - (IP-сеть, subnet) - множество комп-ов, у которых старшая часть IP-адреса одинаковая (октет - чать IP-адреса, отделенная точками).

Структура IP-адреса:

- номер подсети - старшие биты
- номер хоста (комп-а в сети) - младшие биты

Пример: 213.180.193.3 = 213.180.193 (номер подсети) + 0.0.0.3 (номер хоста)

Маска подсети - показывает, где в IP-адресе номер подсети, а где - хоста.

Структура маски:

- длина 32 бита
- единицы в позициях, задающих номер сети
- нули в позициях, задающих номер хоста

IP-адрес (дес.): 213.180.193.3

IP-адрес: 11010101.10110100.11000001.00000011

Маска: 11111111.11111111.11111111.00000000

Подсеть (через логическое И, т.е. AND): 11010101.10110100.11000001.00000000

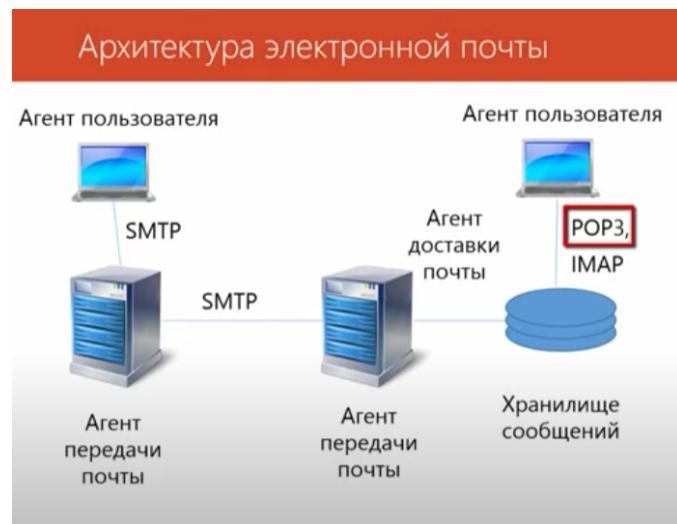
Подсеть (дес.): 213.180.193.0

Маска подсети в виде префикса (ск-ко бит - номер подсети, остаток - номер хоста):
213.180.193.3/24

Типы IP-адресов: индивидуальный (unicast), групповой (multicast) и широковещательный (broadcast, есть ограниченное и направленное, на конце адреса - кол-во всех битов).

Работа эл. почты

В почтовом адресе используется доменное имя, для его определения почтовый сервис взаимодействует с DNS, используя MX записи. Чтобы посмотреть записи MX для домена можно использовать утилиту nslookup -type=mx gmail.com



Flask

Подробнее: <https://flask.palletsprojects.com/en/2.1.x/installation/#python-version>

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Tkinter GUI

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-tkinter>

Tkinter modules

Паттерны программирования

Команды:

- ls - посмотреть, в какой сейчас директории
- cd. /name - перейти в файл

Базы данных и Python

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-database-connectivity>

Git

Шпаргалка: <https://github.com/cyberspacedk/Git-commands>

Вопрос-ответ

- **Что такое NaN ?**

Nan - Not a Number - используется для представления записей, которые не определены, а т.ж. отсутствующих значений в наборе данных.

- **Что такое var?**

▼ **Как получить все аргументы функции (включая те, что не объявлены, но все-таки были переданы)?**

```
import inspect

def pair(boy, girl):
    print(boy+" and "+girl+" are a nice couple.")

pair(boy="John", girl="Lily")
a = inspect.getfullargspec(pair)
print(a)
```

→

John and Lily are a nice couple.

```
FullArgSpec(args=['boy', 'girl'], varargs=None, varkw=None, defaults=None, kwonlyargs=[], kwonlydefaults=None, annotations={})
```

- **Что такое чистая функция?**

Чистая функция — это функция, возвращаемое значение которой зависит только от передаваемых аргументов, без побочных эффектов. Проще говоря, если вы вызываете функцию n раз с n аргументами, и функция всегда возвращает одно и тоже значение, значит, она является чистой

- **Что такое деструктуризация?**

<https://codecamp.ru/blog/python-list-destructuring-aka-packing-and-unpacking/>

- **Что такое ООП? Из чего состоит?**

<https://smartqa.ru/courses/python/lesson-6>

Объектно-ориентированное программирование — это подход, при котором вся программа рассматривается как набор взаимодействующих друг с другом объектов. При этом нам важно знать их характеристики

Инкапсуляция — скрытие поведения объекта внутри него. Объекту «водитель» не нужно знать, что происходит в объекте «машина», чтобы она ехала. Это ключевой принцип ООП.

Наследование. Есть объекты «человек» и «водитель». У них есть явно что-то общее. Наследование позволяет выделить это общее в один объект (в данном случае более общим будет человек), а водителя — определить как человека, но с дополнительными свойствами и/или поведением. Например, у водителя есть водительские права, а у человека их может не быть. (class extends)

Полиморфизм — это переопределение поведения. Можно снова рассмотреть «человека» и «водителя», но теперь добавить «пешехода». Человек умеет как-то передвигаться, но как именно, зависит от того, водитель он или пешеход. То есть у пешехода и водителя схожее поведение, но реализованное по-разному: один перемещается ногами, другой — на машине.

Переопределение каких-то методов у различных классов. К примеру изменения метода `toString()` для конкретного класса

- **Что такое абстракция?**

Абстракция - это способ создания простой модели, которая содержит только важные свойства с точки зрения контекста приложения, из более сложной модели. Иными словами - это способ скрыть детали реализации и показать пользователям только функциональность. Абстракция игнорирует нерелевантные детали и показывает только необходимые. Важно помнить, что мы не можем создать экземпляр абстрактного класса.

Всё программное обеспечение - это абстракция, скрывающая всю тяжелую работу и бездумные детали.

Многие программные процессы повторяются снова и снова. Поэтому, на этапе декомпозиции проблемы, мы удалим дублирование, записывая какой-либо компонент (функцию, модуль, класс и т. Д.), присваивая ему имя (идентификатор) и повторно используя его столько раз, сколько нам нужно.

Процесс декомпозиции - это процесс абстракции. Успешная абстракция подразумевает, что результатом является набор независимо полезных и перекомпонованных компонентов.

- **Какие еще парадигмы знаешь?**

1) Императивный стиль — это парадигма, основанная на составлении алгоритма действий (инструкций/команд), которые изменяют состояние (информацию/данные/память) программы. Фактически, программа на этих языках — это код, который выполняется компьютером сразу, без предварительной компиляции. Из языков высокого уровня, требующих компиляции исходного кода программы в машинный код (или интерпретации), к императивным можно отнести C, C++, Java.

2) Декларативный стиль — это парадигма, при которой описывается желаемый результат, без составления детального алгоритма его получения. В пример можно привести HTML и SQL. При создании HTML мы с помощью тегов описываем, какую хотим получить страничку в браузере, а не то, как нарисовать на экране заголовок статьи, оглавление и текст. В SQL, если нам нужно посчитать количество сотрудников с фамилией «Сидоров», мы напишем `SELECT count(*) FROM employee WHERE last_name = 'Сидоров';`. Тут ничего не сказано про то, в каком файле или области памяти находятся данные по сотрудникам, как именно выбрать из них всех Сидоровых и нужно ли вообще это делать для подсчёта их количества.

Парадигмы декларативного стиля:

Логическое программирование

В целом это скорее математика, чем программирование. Его суть заключается в том, чтобы, используя математические доказательства и законы логики, решать бизнес-задачи. Чтобы использовать логическое программирование, необходимо уметь переводить любую задачу на язык математики. Логическое программирование часто используется для моделирования процессов.

Функциональное программирование

Самая известная парадигма декларативного стиля — функциональное программирование. В этой парадигме понятие функции близко к математическому понятию функции. То есть это штука, которая как-то преобразует входные данные. Особенность функции в этой парадигме в том, что она должна быть чистой, то есть должна зависеть только от аргументов и не может иметь никаких побочных эффектов. Побочный эффект — это какое-либо изменение внешней среды. Если функция меняет глобальную переменную или, например, вызывает метод внешнего объекта, она меняет внешнюю среду. Это и есть побочный эффект.

- **Как проходит инициализация класса и экземпляра класса? Какие есть свойства (поля) класса? Какие есть методы экземпляра класса?**

<https://smartiqa.ru/courses/python/lesson-6> - про классы и объекты

```
class Character:  
    def __init__(self, mana, power):  
        self.m = mana  
        self.p = power  
  
warrior = Character(100, 200)  
print(warrior.m, warrior.p)
```

- **Что геттеры и сеттеры?**
- **Какие есть статичные методы?**
- **Наследование extends**
- **Родительский конструктор: super()**
- **Работа с файлами**
Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-file-io>
- **CSV в Python**
Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-csv>

- **Родительский класс: super**
- **Какие есть способы расширения функциональности класса?**
- **Что такое промисификация?**
- **Чем отличается HTTP2 от HTTP ?**

HTTP/2 (HTTP/2.0) – это бинарный протокол, в отличии от предыдущих версий. Это значит, что данные теперь занимают меньше места и быстрее обрабатываются.

Один из основных плюсов второй версии HTTP. В предыдущей версии для одного запроса была необходима установка отдельного TCP-соединения. И чем больше было запросов, тем медленней работал браузер. Благодаря мультиплексированию браузер отправляет сразу несколько запросов через одно TCP-соединение.

Приоритизация

Первый вид приоритизации подразумевает получение потоком определенного веса, который дальше распределялся между потоками, чтобы нагрузка была равномерной. Данный тип выбора приоритета впервые был применен в протоколе SPDY.

Второй вариант является основным для http/2, а его суть заключается в том, что браузер запрашивает у сервера отдельную загрузку некоторых элементов контента, к примеру, сначала скриптов JavaScript, а затем изображений.

Если сравнивать его с прошлой версией протокола, то здесь разработчики поменяли методы распределения данных на фрагменты и их отправку от сервера к пользователю и наоборот. Новая версия протокола позволяет серверам доставлять информацию, которую клиент пока что не запросил. Это было внедрено с той целью, чтобы сервер сразу же отправлял браузеру для отображения документов дополнительные файлы и избавлял его от необходимости анализировать страницу и самостоятельно запрашивать недостающие файлы.

Еще одно отличие http 2.0 от версии 1.1 – мультиплексирование запросов и ответов для решения проблемы блокировки начала строки, присущей HTTP 1.1. Еще в новом протоколе можно сжимать HTTP заголовки и вводить приоритеты для запросов.

- **Что такое CSRF, XSS?**

CSRF (англ. cross-site request forgery — «межсайтовая подделка запроса», также известна как XSRF) — вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Если жертва заходит на сайт, созданный злоумышленником, от её лица тайно отправляется запрос на другой сервер (например, на сервер платёжной

системы), осуществляющий некую вредоносную операцию (например, перевод денег на счёт злоумышленника). Для осуществления данной атаки жертва должна быть аутентифицирована на том сервере, на который отправляется запрос, и этот запрос не должен требовать какого-либо подтверждения со стороны пользователя, которое не может быть проигнорировано или подделано атакующим скриптом.

XSS (англ. Cross-Site Scripting — «межсайтовый скрипting») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «Внедрение кода».

- **Что такое DOM?**

DOM – это представление HTML-документа в виде дерева тегов.

▼ Пример

```
<!DOCTYPE HTML>
<html>
<head>
<title>О лосях</title>
</head>
<body>
Правда о лосях.
</body>
</html>
```

- **Что такое webApi ?**

- 1). Какое определение можно дать для WEB API и зачем он нужен?

Веб-API - это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения (включая любые используемые веб-фреймворки), и поэтому обычно не включает детали реализации веб-сервера или браузера, такие как SAPI или API, если они не доступны для общего доступа через удаленное веб-приложение.

Серверный веб-API - это программный интерфейс, состоящий из одной или нескольких общедоступных конечных точек для определенной системы сообщений запрос-ответ, обычно выраженной в JSON или XML, которая предоставляется через Интернет - чаще всего посредством HTTP веб сервера.

Гибридные приложения - это веб-приложения, сочетающие в себе использование нескольких серверных веб-API.

Веб-хуки - это серверные веб-API, которые принимают входные данные в виде универсального идентификатора ресурса (URI), который предназначен для использования в качестве удаленного именованного канала или типа обратного вызова, так что сервер действует как клиент для разыменования предоставленного URI и запуска событие на другом сервере, который обрабатывает это событие, тем самым обеспечивая тип однорангового IPC.

2). Можно ли сказать что если сервер на POST или GET запрос возвращает в ответ контент в формате JSON, то это у меня WEB API?

Да

3). Являются ли web-сервисами, например WCF, WEP API?

WCF и ASP.NET Web API - это фреймворк/библиотека с помощью которой вы можете организовать работу WEB-API в вашем приложении.

- **Что такое рекурсия и чем она опасна?**
- **Что будет при обработке тяжелой вычислительной функции (например цикла суммирования) и как решить возникающие проблемы?**
- **Что такое генераторы и итераторы? Что такое yield?**
- Какие есть режимы работы с файлами?

a - добавить что-то в конец файла и создать файл, если его еще нет

w - очищает файл и записывает информацию заново

r - просто чтение (если файла нет - ошибка)

- **Как устроен сборщик мусора в Python?**
Как только объекты больше не нужны, Python автоматически освобождает память из под них. Подробнее: <https://habr.com/ru/post/417215/>.
- **Возможные виды утечек памяти и как с ними бороться?**
- **Про прокси**
- **Что такое SOLID? KISS, DRY, YAGNI?**

SOLID:

- **Single Responsibility Principle** - для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.
- **Open-Closed Principle** - программные сущности ... должны быть открыты для расширения, но закрыты для модификации.
- **Liskov Substitution Principle** - объекты в программе должны быть заменяемыми на экземпляры их подтипов без изменения правильности выполнения программы.
- **Interface Segregation Principle** - много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения.
- **Dependency Inversion Principle** - зависимость на Абстракциях. Нет зависимости на что-то конкретное.

KISS (Keep It Simple Stupid) - утверждает, что большинство систем работают лучше всего, если они остаются простыми, а не усложняются.

DRY (Don't Repeat Yourself) - это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода, особенно в системах со множеством слоёв абстрагирования.

YAGNI (You Aren't Gonna Need It) - процесс и принцип проектирования ПО, при котором в качестве основной цели и/или ценности декларируется отказ от избыточной функциональности, — то есть отказ добавления функциональности, в которой нет непосредственной надобности.

- **Что такое унарный, бинарный, тернарный оператор?**
- **Что такое вычислительная сложность?**
- **Что такое область видимости?**

Обычно, по области видимости, переменные делят на глобальные и локальные. Глобальные существует в течении всего времени выполнения программы, а локальные создаются внутри методов, функций и прочих блоках кода, при этом, после выхода из такого блока переменная удаляется из памяти.

В Python их 4:

- 1). Local - эту область видимости имеют переменные, которые создаются и используются внутри функций.
- 2). Enclosing - суть данной области видимости в том, что внутри функции могут быть

вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в enclosing области видимости.

- 3). Global - Переменные области видимости global – это глобальные переменные уровня модуля (модуль – это файл с расширением .py).
- 4). Built-in - уровень Python интерпретатора. В рамках этой области видимости находятся функции open, len и т.п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. Built-in – это максимально широкая область видимости.

- **Что такое lambda функции (стрелочные функции)?**
- **Что такое мемоизация и каррирование в Python?**

▼ Мемоизация – способ оптимизации при котором сохраняется результат выполнения функции и этот результат используется при следующем вызове. (дело тут, по сути, в быстроте выполнения).

```
from functools import lru_cache #LRU - Least Recently Used
```

```
@clock  
@lru_cache()  
  
def fib(n):  
    if n < 2:  
        return n  
  
    return fib(n-2) + fib(n-1)  
  
print('fib(20) =', fib(20))
```

▼ Каррирование – это преобразование функции от многих аргументов в набор функций, каждая из которых является функцией от одного аргумента. Мы можем передать часть аргументов в функцию и получить обратно функцию, ожидающую остальные аргументы.

```
def greet_deepliy_curried(greeting):  
  
    def w_separator(separator):  
        def w_emphasis(emphasis):  
            def w_name(name):  
                print(greeting + separator + name + emphasis)  
  
            return w_name  
  
        return w_emphasis  
  
    return w_separator
```

```
    return w_name

    return w_emphasis

return w_separator

greet = greet_deepliy_curried("Hello")("...")(".")

greet('German')
greet('Ivan')
```

————через lambda————

```
greet_deepliy_curried =lambda greeting: lambda separator: lambda emphasis:
    lambda name: \
        print(greeting + separator + name + emphasis)
```

- **Что такое менеджер контекста в Python?**
- **Что такое частичное применение функции в Python?**

▼ Это процесс применения функции к части ее аргументов. Другими словами, функция, которая принимает функцию с несколькими параметрами и возвращает функцию с меньшим количеством параметров. Подробнее:
<https://habr.com/ru/post/335866/>.

```
from functools import partial

def greet(greeting, separator, emphasis, name):
    print(greeting + separator + name + emphasis)

newfunc = partial(greet, greeting='Hello', separator=',', emphasis='.')
newfunc(name='German')
newfunc(name='Ivan')
```

- **Что такое замыкание в Python?**

Замыкание (*closure*) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами. Подробнее:
<https://devpractice.ru/closures-in-python/>.

▼ Пример

```
>>> def fun1(a):
    x = a * 3
    def fun2(b):
        nonlocal x
        return b + x
    return fun2

>>> test_fun = fun1(4)

>>> test_fun(7)
19
```

- **Что такое поверхностное и глубокое копирование в Python?**

▼ Поверхностное (**shallow**) копирование - также создает отдельный новый объект или список, но вместо копирования дочерних элементов в новый объект, оно просто копирует ссылки на их адреса памяти. Следовательно, если вы сделаете изменение в исходном объекте, оно будет отражено в скопированном объекте, и наоборот.

```
# Program 2 - Shallow Copy

import copy

result_A = [95, 85, 82]

result_B = copy.copy(result_A)

print(result_A)

print(result_B)
```

▼ Глубокое (**deep**) копирование - создает новую и отдельную (независимую) копию всего объекта или списка со своим уникальным адресом памяти.

```
# Program 1 - Deep Copy

import copy

result_A = [90, 85, 82] # Student A grades

result_B = copy.deepcopy(result_A) # Student B grades (copied from A)

print(result_A)

print(result_B)
```

- **Что такое магические методы класса в Python?**

- **Что такое деструктуризация?**

Деструктуризация - распаковка, т.е. разбиение целого итерабельного объекта(например, списка) на части. Подробнее: <https://codecamp.ru/blog/python-list-destructuring-aka-packing-and-unpacking/>.

- **Что такое функциональное программирование в Python?**

- **Что такое магические методы в Python?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-magic-methods>

- **Какие есть ключевые слова в Python?**

- **Что такое регулярные выражения?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-regex>

- **Как реализуется асинхронность в Python?**

- **Как Python взаимодействует с REST API?**

- **Что такое модуль mimetypes в Python?**

- **Как происходит обработка событий в Python?**

- **Что такое worker в Python?**

- **Что такое стек вызовов в Python?**

- **Хранение данных в backend**

- **Как работают токены в Python?**

- **Что такое унарный, бинарный, тернарный операторы Python?**

- **Что такая вычислительная сложность в Python?**

- **Как работает обработка ошибок и исключений в Python?**

- **Что такое polyfill в Python?**

- **Что такое socket module?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-socket-module>

▼ Вопросы на собеседовании

1. В чем разница между модулем и пакетом в Python?

- 2. Какие встроенные типы доступны в Python?**
- 3. Что такое лямбда-функция в Python?**
- 4. Что означает пространство имен?**
- 5. Объясните разницу между списком и кортежем?**
- 6. Чем отличается pickling от unpickling?**
- 7. Что такое декораторы в Python?**
- 8. Разница между генераторами и итераторами?**
- 9. Как преобразовать число в строку?**
- 10. Как используется оператор // в Python?**
- 11. Есть ли в Python инструкция Switch или Case, как в C?**
- 12. Что такое функция range() и каковы ее параметры?**
- 13. Как используется %s?**
- 14. Обязательно ли функция Python должна возвращать значение?**
- 15. Есть ли в Python функция main()?**
- 16. Что такое GIL?**
- 17. Какой метод использовался до оператора «in» для проверки наличия ключа в словаре?**
- 18. Как изменить тип данных списка?**
- 19. Каковы ключевые особенности Python?**
- 20. Объясните управление памятью в Python**
- 21. Что такое PYTHONPATH?**
- 22. Чувствителен ли Python к регистру?**
- 23. Объясните использование функций `help()` и `dir()`.**
- 24. Что такое модули Python?**
- 25. Объясните, что означает «self» в Python.**
- 26. Что такое инженерия и процесс разработки в целом?**

27. Какие знаете принципы программирования?
28. Чем отличаются процедурная и объектов-ориентированная парадигмы программирования?
29. Какие основные принципы ООП (наследование, инкапсуляция, полиморфизм)?
30. Что такое множественное наследование?
31. Какие есть шесть этапов разработки продукта в Software Development lifecycle и какая разница между Agile и Kanban?
32. Какие есть методы HTTP-запросов и какая между ними разница?
33. Как выглядят HTTP-request / response?
34. Что такое авторизация и как она работает?
35. Что такое cookies?
36. Что такое веб уязвимость?
37. Какие знаете классические базы данных?
38. Как читать спецификацию в конкретном языке (например, PEP8 в Python)?
39. Как происходит взаимодействие клиента и сервера?
40. Какие есть подходы к проектированию API?
41. Как использовать паттерны программирования?
42. Что такое Acceptance Testing и зачем его используют?
43. Что такое модульные и интеграционные тесты, API-тесты?
44. Как писать unit-тесты?
45. Какие есть best practices в написании автотестов?
46. Какие базовые команды системы контроля версий?
47. Как использовать Git?
48. В чем разница между хешированием и шифрованием?
49. Python - интерпретируемый язык или компилируемый?
50. Какие есть меняющиеся и постоянные типы данных?

51. Что такое область видимости переменных?
52. Что такое introspection?
53. Разница между `is` и `==`?
54. Разница между `__init__()` и `__new__()`?
55. В чем разница между потоками и процессами?
56. Какие есть виды импорта?
57. Что такое класс, итератор, генератор?
58. Что такое метакласс, переменная цикла?
59. В чем разница между итераторами и генераторами?
60. В чем разница между `staticmethod` и `classmethod`?
61. Как работают декораторы, контекстные менеджеры?
62. Как работают `dict comprehension`, `list comprehension` и `set comprehension`?
63. Можно ли использовать несколько декораторов для одной функции?
64. Можно ли создать декоратор из класса?
65. Какие есть основные популярные пакеты (`requests`, `pytest`, etc)?
66. Что такое lambda-функции?
67. Что означает `*args`, `**kwargs` и как они используются?
68. Что такое `exceptions`, `<try-except>`?
69. Что такое PEP (Python Enhancement Proposal), какие из них знаете (PEP 8, PEP 484)?
70. Напишите hello-world сервис, используя один из фреймворков.
71. Какие есть типы данных и какая разница между `list` и `tuple`, зачем они?
72. Как использовать встроенные коллекции (`list`, `set`, `dictionary`)?
73. В чем заключается сложность доступа к элементам `dict`?
74. Как создается объект в Python, для чего `new`, зачем `init`?
75. Что знаете из модуля `collections`, какими еще `built-in` модулями пользовались?

76. Что такое шаблонизатор и как в нем выполнять базовые операции (объединять участки шаблона, выводить дату, выводить данные с серверной стороны)?
77. Как Python работает с HTTP-сервером?
78. Что происходит, когда создается виртуальная среда?
79. Какие есть базовые методы работы с SQL-базой данных в Python?
80. Что такое SQL-транзакция?
81. Как сделать выборку из SQL-базы с простой агрегацией?
82. Как выглядит запрос, который выполняет JOIN между таблицами и к самим себе?
83. Как отправлять запросы в SQL-базу данных без ORM?
84. Что такое алгоритмы (например, Big-O notation)?
85. Какие есть базовые алгоритмы сортировки?
86. Что такое Bubble Sort и как это работает?
87. Что такая линейная сложность сортировки?
88. Ориентируетесь ли в *nix, можете ли написать скрипты/автоматизацию для себя и коллег?
89. Что такое многопоточность?
90. Что такое архитектура веб сервисов?
91. Как работает современное нагруженное веб приложение (нарисовать и обсудить примерную архитектуру, например, Twitter или Instagram)?
92. Что нужно для сайта / сервиса среднего размера (redis \ celery \ кэш \ логирование \ метрики)?
93. Как написать, задеплоить и поддерживать (микро) сервис?
94. Как масштабировать API?
95. Як проводить Code review?
96. Что такое абстрактная фабрика, как ее реализовать и зачем ее применяют?
97. Что такая цикломатическая сложность?
98. Async Python: как работает, зачем, что под капотом?

99. Сравнить асинхронные web-фреймворки.
100. Что такое модель памяти Python?
101. Что такое SQLAlchemy (Core и ORM частей) и какие есть альтернативы?
102. Принципы работы и механизм Garbage collection, reference counting?
103. Как работает thread locals?
104. Что такое *slots*?
105. Как передаются аргументы функций в Python (by value or reference)?
106. Что такое type annotation?
107. Для чего используют нижние подчеркивания в именах классов?
108. Статические анализаторы: Flake8, Pylint, Radon.
109. Разница между SQL и NoSQL?
110. Как оптимизировать SQL-запросы?
111. Какие есть уровни изоляции транзакций?
112. Какие есть виды индексов?
113. Точечные вопросы по выбору БД, движков БД?
114. **Front-end:** есть ли опыт работы с «современным» JS (Babel, Webpack, TS, ES)?
115. **DevOps:** работали ли с Docker-контейнерами, объяснить основные термины K8s (кластер, pod, node, deployment, service), что такое Kibana?
116. **Алгоритмы:** что такое времененная сложность алгоритма (time complexity)?
117. **Углубленные знания Linux:** как зайти на внешний сервер, работать с пакетами, настроить среду и выполнять операции?
118. **Специфично для Data Science:** как работать с пакетами для обработки и визуализации данных (NumPy, Pandas и другие)?
119. Что такое @property?
120. Каким образом можно запустить код на Python параллельно?
121. Как работать с stdlib?

122. Какие задачи решали с помощью метаклассов?
123. Что такое дескрипторы?
124. Знания других языков, кроме Python (опыт).
125. Какие технологические особенности реализации распределенных систем?
126. Какие есть низкоуровневые особенности языков и фреймворков?
127. Способы и методы управления памятью.

Загуглить:

- MongoDB
- Redis
- Tarantool
- Kafka
- декомпозиция
- веб-сервисы
- сетевые протоколы
- ui-фреймворки
-

1. Основы (как работает интернет):

Как работает Интернет?

Центр обработки данных (например, гугловый) м. б. в тысячах км от хоста – на нем хранятся данные.

Сеть волоконно-оптических кабелей охватывает весь мир и соединяют центр обработки данных и конкретный хост.

Мобильная сеть или маршрутизатор вай-фай – посредники между хостом и оптоволоконной сетью.

Конкретный файл, который необходимо передать хранится в центре обработки данных, а точнее на твердотельном накопителе (SSD), который выполняет функции внутренней памяти сервера.

Каждое устройство, подключенное к сети, имеет IP-адрес (его присваивает устройству интернет-провайдер). Кстати, сервер в ЦОД также имеет IP-адрес (если его узнать, сайты, которые на нем хранятся станут доступны).

IP-адресу соответствует доменное имя для удобства запоминания пользователем.

Сервер хранит несколько сайтов одновременно. однако они не могут быть привязаны к одному IP-адресу. Для разрешения этого неудобства используются заголовки хоста.

Чтобы серверу легче было соотнести IP-адрес с его доменным именем существует DNS-сервер.

DNS-сервером могут управлять интернет провайдеры и др. компании.

Сайт состоит из файлов, которые хранятся на сервере.

Как проходит поиск сайта:

1. Вводим в адресной строке домен
2. Браузер отправляет запрос на DNS-сервер
3. DNS-сервер ищет соответствующий IP-адрес и отдает его браузеру
4. Браузер перенаправляет запрос с IP-адресом в ЦОД (т.е. к серверу, на котором хранится сайт)
5. Сервер получает запрос на доступ к сайту, и начинается поток данных. Они передаются в цифровом формате (последовательности 0 и 1, которые разделены на пакеты, каждый по 6 битов, в каждом из которых также есть порядковый номер и IP-адрес хоста) через оптоволоконный кабель в виде световых импульсов.
6. Маршрутизатор (модем, например) преобразует световые сигналы оптоволоконного кабеля в электрические. А для передачи эл. сигналов на хост. ноутбук, в частности, используется кабель Ethernet, а с мобильной связью сигналы из оптоволокна направляются на вышку сотовой связи, с которой сигнал в виде электромагнитных волн передается на смартфон.
7. Браузер (он же клиент) получает данные и отрисовывает веб-страницу

Организация ICANN занимается регистрацией доменных имен, контролем IP-адресов и т.д.

Для скорой передачи данных каждый пакет выбирает кратчайший доступный маршрут, а там собираются в соответствии с порядковыми номерами, если часть информации не достигла хоста, с него отправляется повторный запрос на отправку данных.

Для управления потоками данных существуют протоколы (http/https – веб доступ, TCP/IP – передача данных, RTP – медиа, т.е. онлайн-видео, онлайн стримы, IP-телефония). Для разных приложений протоколы различны (исходя из нужд).

Frontend - клиентская часть сайта (та, которую мы видим, браузер, для ее создания используются CSS – каскад стилей для HTML, HTML - разметка, JavaScript). Backend – часть, связанная с сервером БД (общение происходит посредством серверных языков программирования – JavaScript, Python, Ruby, PHP, SQL)

MAC-адрес (media access control) – правила именования устройств на канальном уровне (для устройств: сетевых плат и т.д.). Называет их компания-изготовитель.

К IP-адресу добавили битовую маску. Есть биты, логические операции (пример & - если и то и то True, т.е. 1). IP-адрес состоит из 4 байт (IPv4), каждый из 8 бит. После применения операции – другое число, чтобы определить, какие биты исп. для нумерации сети и узлов внутри сети. Пр.: число до – к сети, после – к узлу.

У узла есть ИП и фикс. маска подсети. Для него внутренние адреса – номер сети ИП совпадает – узлы локальной сети, непосредственно доступны. Внешние узлы – через шлюз/маршрутизатор (у него один адрес в одной сети, а другой – в другой).

Таблица маршрутизации: номер сети назначения (IP через маску), IP-адрес шлюза, метрика.

Как работает ОС

Процессор:

- Читает инфу из памяти
- Выполняет операцию по инструкции
- Кладет в память результат

BIOS – basic input-output system - устройство ввода/вывода - первый интерфейс. Т.е. у компа теперь 2 режима: устройство вычислений и устройство ввода/вывода (делает удобней).

ОС отделяет приложения друг от друга и от себя, чб при поломке одного остальные не страдали.

Позволяет получить абстракции, облегчающие работу.

Абстракции:

- процессы и потоки
- файлы и файловые системы
- адресное пространство и память
- сокеты, протоколы, устройства и т.д.

Интерфейс системных вызовов

Kernal Space (OS) (– вызовы выполняются внутри ядра) и User Space (- а вызываются снаружи) – разграничивает процессор.

BASH – интерпретатор командной строки.

Ядро:

- обрабатывает запросы приложений
- обрабатывает запросы оборудования
- обеспечивает диспетчеризацию процессов (scheduling)
- обрабатывает исключительные ситуации

Сервисы, предоставляемые ОС:

- управление процессами
- управление памятью
- файлы, их содержимое, каталоги и директории
- модель безопасности
- межпроцессное взаимодействие
- прочее: терминалы, сети, таймеры, периферия

Драйвер – превращение интерфейса программы в общий интерфейс

Процесс – абстракция, которая предоставляет иллюзию ПК (как будто все ресурсы направлены на пользователя или процесс)

- адресное пространство (страницы, таблицы)
- CPU
- файлы и др. абстракции ОС
- состояние (состояние в памяти и набор регистров внутри процессора)

- стек

Поток – поток исполнения инструкций, процесса или его части

TGID – про поток

PID – про процесс

Вытесняющая многозадачность

Прерывание – ситуация остановки процессором последовательного выполнения программы для выполнения запроса или реакции на событие.

Системный вызов – специальное программное прерывание, соответствующее запросу сервера у ядра.

Иключение – неверное действие программы, приводящее к генерации прерывания.

Системные вызовы и драйверы

System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n bytes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

Многозадачность:

- прерывание таймера
- смена контекста
- план блокировок (при наличии нескольких CPU)
- освобождение ресурсов при завершении процесса

состояние процессора хранится в регистрах

машинные коды – атомарные, их не видно среднестатистическому прогеру

deadlock – когда есть 2 критических процесса, которые происходят в критической секции и задействуют критические ресурсы

Active – состояние, когда процесс выполняется на процессоре

Waiting – состояние, когда процесс вызывает операцию ввода-вывода, которое не может сразу выполнено

Ready - когда операция завершилась, готова к исполнению, но не выполнится

Физическая память

Виртуальная память (пейджинг, страничная организация памяти)

Процессор:

x86 – архитектура в большинстве компьютеров (32 и 64 бита)

ARM - архитектура в мобильных устройствах

AVR – авто, телевизоры и т.д.

Для каждой архитектуры пишется ПО.

Разрядность – величина, которая определяет размерность машинного слова (макс. кол-во бит, к-ми может оперировать процессор за раз)

Команду процессор считывает из памяти и кладет куда-то. Регистровая память (временные результаты вычислений, быстрая) и кэш память. Максимальный размер регистра = разрядности.

Регистр д.б. кратен ячейке ОЗУ. Байтовая адресация – каждая ячейка = 1 байту и каждый байт имеет свой адрес, по которому процессор может к нему обратиться. Словесная адресация – то же самое, но размер ячейки = машинному слову, и процессор обр-ся к машинному слову.

ASCII-символ = 7 бит

Регистры специального назначения – предназначаются для конкретного содержимого. В сегментных регистрах хранятся адреса памяти; регистры для работы со стеком указывают на каналы фрейма и верхушку стека; флаговые регистры содержат различные биты, которые отражают состояние результата предыдущей операции; указатель команд (instruction pointer) указывает адрес команды, которую нужно выполнить следующей и др.

Регистры общего назначения – могут быть использованы на усмотрение прогера, однако есть опр. соглашения по работе компиляторов. Выступают в роли переменных, параметров, временного хранилища для результатов вычислений.

Язык ассемблера – низкоуровневый язык программирования, состоящий из символического обозначения машинных команд.

Ассемблер – программа-транслятор этого языка в машинный код.

При включении компьютер запускает биос, его задачи: найти первое дисковое устройство, которое было указано, взять оттуда первый сектор, загрузить его в память и передать на него управление (указать процессору на то, чтобы теперь он начал выполнять команды по адресу этой загруженной программы, а именно - ОС). ОС переводит процессор в безопасный режим. ОС выставляет специальный флаг системного регистра, устанавливает разрешения и условия для других программ, распределяет таблицы дескрипторов и прерываний. Т.о. ОС становится полноценным хозяином компа. Далее реализуется работа с памятью, выставляются различные ограничения и т.д.

Когда мы запускаем программу, ОС выделяет под нее место, загружает ее в ОЗУ (например, жесткого диска), после чего передает ей управление, т.е. говорит процессору, с какого места из памяти ему нужно выполнить следующую команду. На этом моменте используется регистр IP. После процессор обновляет значение в этом регистре, чб он указывал на следующую инструкцию в памяти. Т.е. он определяет размер инструкции и прибавляет его к значению в IP.

Режимы работы процессора:

- реальных адресов – 16-битный режим, в который процессор переходит сразу после включения компьютера (адрес, сформированный программами, я-ся реальными не требует преобразований)
- защищенный – 32-битный режим, в который можно перейти только из режима реальных адресов, при нем обеспечивается защита данных, ОС, прикладных программ и данных этих программ друг от друга благодаря разделению приложений на разные уровни привилегий
- 64 разрядный режим – в него можно перейти только из защищенного режима

В оф. док-ии:

- режим реальных адресов + защищенный – legacy mode
- 64 разрядный + режим совместимости (поддержка 32 и 16-разрядного кода) – long mode

Уровни привилегий – доступ к использованию ресурсов процессора (начинаются с защищенного режима).

0 уровень – полный доступ к процессору (на нем ОС)

- 1 уровень – на нем дей-т запреты с 0 уровня
- 2 уровень – запреты с 0 и 1 ур.
- 3 уровень – (пользовательский, на нем прикладные проги) запреты с 0, 1 и 2 ур.

Устройство адресации. Вся оперативная память поделена на сегменты. Их размер зависит от режима, в котором работает процессор. Ячейки представляются в специальном формате: логический адрес = адрес начала сегмента (16 бит) : смещение в сегменте (16 бит).

В режиме реальных адресов адрес делится на сегменты по 64 кб.

Логический адрес преобразуется в 20 битный адрес ячейки памяти: физический адрес = адрес начала сегмента << 4 + смещение сегмента.

Максимальный размер адресов – 1мб физической памяти.

В защищенном режиме память делится на сегменты от 0 до 4 гб. ОС создает иллюзию того, что каждой программе доступно все пространство памяти. Она активирует механизм трансляции виртуальных адресов в физические, т.е. программы начинают работать в виртуальном адресном пространстве, про которое она сами не догадывается. Они продолжают формировать логические адреса, как и раньше, предполагая, что обращаются к реальной памяти.

Логический адрес, созданный программой в этом режиме, преобразуется не в физический, а в виртуальный: логический адрес = селектор дескриптора (16 бит) + смещение в сегменте (32 бита).

Селектор дескриптора – индекс дескриптора: уровень привилегий + таблица + индекс. Таблица и индекс показывают GDT/LDT (начальный адрес сегмента, уровень привилегий, размер сегмента).

Виртуальный адрес (32 бита) = адрес начала сегмента + смещение в сегменте.

Страницчная организация памяти – структура, представляющая собой виртуальную память. Все адресное пространство разбивается на страницы, чей размер зависит от режима работы процессора и режима трансляции адресов. Каждая страница описывается структурой, которая объединяется в таблицу страниц, а таблицы страниц объединяются в каталог страниц. Виртуальный адрес = индекс в каталоге страниц + индекс в таблице страниц + смещение в странице.

При помощи механизма трансляции адресов виртуальный адрес преобразуется в физический (32 бита), максимальное число – 4 гб физической памяти.

В 64-битном режиме в основном так же, но есть различия, например, практически полностью отключена сегментация. В физической памяти доступно до 2^{52} байт, в виртуальной – до 2^{48} в виртуальной.

Ограничения обусловлены архитектурой процессора и ОС.

Как внешние устройства взаимодействуют с процессором, если он всегда занят своей работой?

Например, нам нужно, чтобы процессор обработал запрос пользователя с клавиатуры прямо во время работы программы и без задержек. Внешнее устройство (клавиатура) через контроллер прерываний передает процессору сигнал о прерывании, чтобы он прервал выполнение текущей программы и передал управление специальной функции – обработчику прерываний. Конечно, сначала сохраняется состояние текущей программы (регистры, адрес сл. инструкции и т.д.). После совершения работы обработчик передает управление программе, восстанавливая ее последнее сохраненное состояние, и она продолжает работу с того места, на котором была прервана.

Прерывания находятся в специальной таблице дескрипторов прерываний, у которых есть уровни привилегий, которые определяет уровень привилегий программы, которая определяет прерывание.

Виды прерываний:

- программные – генерирует сам программист-оператор в ряде ОС (в винде нельзя, есть спец. проги)
- аппаратные – генерируются контроллером прерываний, которые выполняются согласно приоритетам
- исключения – их генерирует сам процессор при попытке программы нарушить ограничения защиты или при возникновении ошибок в ходе ее выполнения

Механизм многозадачности основан на прерывании – одновременная работа многих программ достигается путем попеременного их переключения.

При реализации ОС программы разделяются на процессы, которые разделяются на потоки/задачи (задача – самостоятельная последовательность команд, которая выполняется в своем окружении).

Многопроцессорность. Каждый процессор оснащен несколькими ядрами, а серверы оснащены несколькими процессорами.

Типы многопроцессорных систем:

- на материнской плате несколько сокетов для процессоров
- когда процессор имеет несколько ядер
- когда процессор имеет виртуальные ядра

Frontend

клиентская часть

Пользовательский интерфейс для общения с сервером. UI/UX

Пул компетенций: HTML, CSS, JS, Git, WebPack, Gulp, SASS, LESS, JQuery, React, View, базовые навыки PS, Figma, Avacode и т.д.

Верстальщик – макеты дизайна, HTML, CSS (логика, JS и прочее).

Backend

программно-аппаратная часть

Хостинг

Это услуга, предоставляемая компаниями, заключающаяся в предоставлении определенному домену доступа к серверу, на котором будет запущено ПО, необходимое для обработки запросов к хранимым файлам (вэб-сервер). Как правило, в услугу входит предоставление места для постовой корреспонденции, баз данных, DNS, файлового хранилища на специально выделенном файл-сервере и т.п., а также поддержка функционирования соответствующих серверов.

Один из главных критериев выбора хостинга – операционная система, т.к. от нее зависит ПО, которое будет поддерживать функциональность сервисов.

Прочие критерии:

- поддержка CGI (стандарт интерфейса, используемого внешней программой для связи с веб-сервисом, или по-другому «скрипты»): Python, Ruby, PHP, Perl, ASP, JSP, Java.
- поддержка .htaccess / .htpasswd – для HTTP-сервиса Apache
- поддержка баз данных, а т.ж. установленные модули и фреймворки для каждой из возможностей

Как услуга хостинг имеет такие критерии, как:

- размер дискового пространства под файлы пользователя (память)
- количество месячного трафика
- количество сайтов, которые можно разместить с одной учетки
- кол-во FTP пользователей
- кол-во e-mail ящиков, пространство под почту
- кол-во баз данных и пространство под них
- кол-во одновременных процессов на пользователя
- кол-во ОЗУ и максимальное время исполнения на каждый процесс пользователя

Качественные показатели:

- свободные ресурсы CPU (центральный процессор), оперативной памяти, которые влияют на быстродействие сервера
- пропускная способность каналов, от которой зависит загрузка информации
- удаленность оборудования хостера от ЦА, которая влияет на скорость загрузки информации

Виды хостинга:

- виртуальный хостинг – сервер с множеством сайтов, владельцы которых имеют одинаковые права и обязанности
- виртуальный выделенный сервер (VPS/VDS) – автономная (выделенная часть дискового пространства на сервере и фиксированные ресурсы. Владелец получает права админа и может сам устанавливать и настраивать программы)
- выделенный сервер - полное владение сервером с отдельной ОС, ПО
- colocation – размещение сервера, которым владеет отдельный человек или компания, в data-центре хостинговой компании.

Сервер ([аппаратное обеспечение](#))

Это выделенный или специализированный компьютер (ПК или рабочая станция) для выполнения сервисного ПО (в т.ч. серверов тех или иных задач).

Это компьютер, выполняющий серверные задачи, или компьютер (или иное аппаратное обеспечение), специализированный (по форм-фактору и/или ресурсам) для использования в качестве аппаратной базы для серверов услуг (иногда — услуг определённого направления), разделяя ресурсы компьютера с программами, запускаемыми пользователем. Такой режим работы называется «невыделенным», в отличие от «выделенного» (англ. dedicated), когда компьютер выполняет только сервисные функции. Строго говоря, на рабочей станции (для примера, под управлением Windows XP) и без того всегда работает несколько серверов — сервер удалённого доступа (терминальный сервер), сервер удалённого доступа к файловой системе и системе печати и прочие удалённые и внутренние серверы.

Сервер ([ПО](#))

Это программный компонент вычислительной системы, выполняющий сервисные функции по запросу клиента, предоставляя ему доступ к определенным ресурсам и услугам.

Для взаимодействия с клиентом (-ами) сервер выделяет необходимые ресурсы межпроцессного взаимодействия и ожидает запросы на открытие соединения. В зависимости от процесса сервер может обслуживать запросы одной системы или на других машинах через каналы передачи данных (пр.: COM-порт) или сетевые соединения.

Формат запросов клиента и ответов сервера определяется протоколом. Спецификации открытых протоколов описываются открытыми стандартами (пр.: документы RFC определяют протоколы Интернета).

Классификация стандартных серверов по типу услуг:

- Универсальные – не выполняют услуг самостоятельно, они предоставляют серверам услуг упрощенный интерфейс к ресурсам межпроцессного взаимодействия и/или унифицированный доступ клиентов к услугам.
 - inetd (internet super-server daemon – демон сервисов IP) – стандартное средство UNIX-систем – программа, позволяющая писать серверы TCP/IP (и сетевых протоколов других семейств), работающие с клиентом через перенаправленные inetd потоки стандартного ввода и вывода (stdin и stdout).
 - RPC (Remote Procedure Call – удаленный вызов процедур) – система интеграции серверов в виде процедур, доступных для вызова удаленным пользователем через унифицированный интерфейс. Сейчас в большинстве UNIX-систем и Windows используется унифицированный интерфейс от Sun Microsystems, изобретенный для их ОС (SunOS, Solaris, Unix-система).
 - Прикладные клиент-серверные технологии Windows:
 - (D-)COM – (Distributed) Component Object Model – модель составных объектов – позволяет одним программам совершать операции над объектами данных, используя процедуры других программ. Изначально предназначена для внедрения и связывания объектов (OLE – Object Linking and Embedding), но в общем позволяет писать широкий спектр различных прикладных серверов. COM – работает в пределах одного компьютера, DCOM – доступна удаленно через RPC.
 - Active-X – расширение COM и DCOM для создания мультимедийных приложений.

Универсальные серверы часто используются для написания информационных серверов – тех, что не нуждаются в специфической работе с сетью и не имеющих никаких задач, кроме обслуживания клиентов (пр.: обычные консольные программы и скрипты могут выступать в роли серверов для inetd).

Большинство внутренних и сетевых специфических серверов Windows работают через универсальные серверы (RPC, (D-)Com).

- Маршрутизация – не совсем сервер, скорее базовая функция поддержки сети операционной системой.

Для TCP/IP маршрутизация является базовой функцией стека IP (кода поддержки TCP/IP). Каждая система в сети выполняет маршрутизацию своих пакетов к месту назначения. Маршрутизаторы (роутеры или шлюзы) выполняют маршрутизацию чужих пакетов (форвардинг), управляемые через таблицы маршрутов и правила, их задачи:

- принять пакет
- найти машину, которой предназначался пакет, или следующий в цепи маршрутизатор
- передать пакет или вернуть ICMP-сообщение о невозможности доставки пакета по причинам:
 - назначение недостижимо (destination unreachable) – у пакета кончилось «время жизни» прежде, чем он был доставлен
 - хост недостижим (host un.) – компьютер или сл. маршрутизатор выключен или не существует

- сеть недостижима (network un.) – маршрутизатор не имеет маршрута в сеть назначения.
 - если пакет не может быть доставлен по причине чрезмерной загрузки маршрутизатора или сети – отбросить пакет без уведомлений.
- Динамическая маршрутизация – имеет своей задачей сбор информации о текущем состоянии сложной сети и поддержание таблицы маршрутов через эту сеть, чтобы обеспечить доставку пакета по кратчайшему и самом эффективному маршруту.

Из решений динамической маршрутизации клиент-серверную модель использует только BGP (Border Gateway Protocol – протокол пограничного шлюза), применяемый для глобальной маршрутизации.

Локальные решения (RIP (протокол маршрутной информации – простейший протокол, позволяющий маршрутизаторам динамически обновлять маршрутную информацию по данным от соседних маршрутизаторов небольшой сети), OSPF (ПДМ. позволяющий отслеживать состояние канала и использующий для нахождения кратчайшего пути алгоритм Дейкстры)) используют в своей работе бродкастовые (широковещательный канал - поток данных предназначен для всех участников сети) и мультикастовые (мультивещание, много адресное – адрес сетевого назначения – мультикастная группа) рассылки.

- Сетевые службы – обеспечивают функционирование сети (пр.: серверы DHCP (прикладной протокол, позволяющий сетевым устройствам автоматически получать IP-адрес и др. параметры, необходимые для работы в сети TCP/IP) и BOOTP (прикладной протокол, позволяющий клиенту автоматически получать IP-адрес, обычно при загрузке компьютера) обеспечивают стартовую инициализацию серверов и рабочих станций, а DNS – трансляцию имен в адреса и наоборот).

Серверы туннелирования (пр.: VPN) и прокси-серверы обеспечивают связь с сетью, недоступной роутингом.

Серверы AAA и Radius обеспечивают в сети единую аутентификацию, авторизацию и ведение логов доступа.

- Информационные службы – к ним относятся как простейшие серверы, сообщающие информацию о хосте (time, daytime, motd) и пользователях (finger, ident (протокол, описывающий способ идентификации пользователя для конкретного соединения TCP)), так и серверы для мониторинга (пр.: SNMP – стандартный интернет-протокол для управления устройствами в IP-сетях на основе архитектур TCP/UDP; поддерживаемые устройства – маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, модемные стойки и др.). Большинство информационных служб работают через универсальные серверы.

Особый вид информ. служб – сервисы синхронизации времени – NTP (сетевой протокол, используемый для синхронизации часов на компьютере с помощью сетей с переменной латентностью). Помимо установки времени на компьютере, NTP периодически опрашивает другие серверы для сверки точности собственного времени, а т.ж. скорости хода часов путем замедления или ускорения.

- Файловые серверы – обеспечивают доступ к файлам на диске сервера. Прежде всего они передают файлы по протоколам:
 - FTP -
 - TFTP – используется в основном при первоначальной загрузке бездисковых рабочих станций, в отличие от FTP не содержит возможности аутентификации и основан на транспортном протоколе UDP.
 - SFTP – протокол прикладного уровня, предназначенный для копирования и выполнения других операций с файлами поверх надежного и безопасного соединения.

- HTTP – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML (текстовые данные), в наст. вр. используется для передачи произвольных данных (веб-страницы, картинки, музыка и т.д.).

Другие серверы позволяют монтировать дисковые разделы сервера в дисковое пространство клиента и полноценно работать с файлами на них. Например, серверы протоколов NFS (протокол сетевого доступа к файловым системам, который позволяет подключать удаленные файловые системы через сеть) и SMB (сетевой протокол прикладного уровня для удаленного доступа к файлам, принтерам и др. сетевым ресурсам, а т.ж. межпроцессного взаимодействия), которые работают через интерфейс RPC.

Недостатки файл-серверной системы:

- большая нагрузка на сеть, высокие требования к пропускной способности (делает практически невозможной одновременную работу большого числа пользователей с большим объемом данных)
 - обработка данных осуществляется на компьютере пользователя (требования к уровню их аппаратного обеспечения)
 - блокировка данных при работе с ними одним пользователем делает невозможной работу с ними других пользователей
 - безопасность, т.к. для работы с файлом придется дать пользователю полный доступ к нему, когда необходима всего часть файла
- Серверы доступа к данным – обслуживают БД и отдают данные по запросу. Один из простейших – LDAP (Lightweight Directory Access Protocol – облегченный протокол доступа к спискам). Для БД единого протокола нет, однако ряд БД объединяет использование единых правил формирования запросов – языка SQL (Structured Query Language – язык структурированных запросов). Остальные БД – NoSQL.
 - Медиа серверы – предоставляют сети доступ к мультимедийным источникам, от аудио/видео по запросу до стриминга в аудио/видео в реальном времени.

- VoIP/ IP-телефония – программные коммутаторы (софтсвитчи), IP-АТС (автоматическая телефонная станция на основе межсетевого протокола IP операторского уровня), виртуальные АТС и серверы ВКС (сервер многочастотной конференции для неск. польз.), а также специализированные серверы Интернет-сервисов (пр.: Skype) обеспечивают пользователей возможностью голосовой и видеосвязи в реальном времени посредством компьютерной сети. Кроме потоковой передачи медиа данных, сервер IP-телефонии подобно классической АТС реализует возможность регистрации окончного терминала, маршрутизацию вызова и корректное установление соединения между пользователями и др.

В отдельных случаях, в зависимости от реализуемой технологии и административных настроек, VoIP-сервер может обеспечивать только управление — регистрацию пользователя в сети и коммутацию поступающих вызовов, без непосредственного участия в передаче медиа-данных между клиентскими терминалами. В этом случае потоковые данные с полезной нагрузкой передаются напрямую между конечными пользователями (peer-to-peer) и / или некоторыми промежуточными устройствами, приложениями. Известно, что такой вариант прямой связи с управлением через сервер применяется в Skype, Viber, Telegram и WhatsApp. Также, подобный режим нередко применяется в корпоративных IP-АТС.

В качестве клиентских терминалов к VoIP-серверу могут выступать VoIP-телефоны, видеотелефоны, программные телефоны (софтфоны), а также обычные аналоговые телефонные аппараты подключенные через VoIP-шлюз. Сервер IP-телефонии может работать как самостоятельное устройство для обеспечения связи между внутренними пользователями или быть подключенным к какой-либо сторонней сети, в том числе к телефонной сети общего пользования, через Интернет или через сеть оператора телефонной связи.

- Службы обмена сообщениями – эл. почты, работающие по протоколу SMTP. SMTP-сервер принимает сообщение и доставляет его в локальный почтовый ящик пользователя или на другой SMTP-сервер (сервер назначения или промежуточный). На многопользовательских компьютерах пользователи работают с почтой прямо на терминале или веб-интерфейсе. Для работы с почтой на ПК почта забирается из почтового ящика через серверы, работающие по протоколам POP3 и IMAP.

Для организации конференций используются серверы новостей, работающие по протоколу NNTP.

Для обмена сообщениями в реальном времени существуют серверы чатов по многочисленным протоколам (пр.: IRC, Jabber (XMPP), OSCAR).

- Серверы удаленного доступа – через соответствующую клиентскую программу обеспечивают пользователя аналогом локального терминала (текстового или графического) для работы на удаленной системе.

Для обеспечения доступа к командной строке служат серверы telnet, RSH, SSH.

В Unix-системах есть встроенный сервер удаленного доступа к графическому интерфейсу – X Window System. В Windows – терминальный сервер.

SNMP-протокол позволяет удаленно производить мониторинг и конфигурацию, для этого на ПК или АУ должен быть SNMP-сервер.

- Серверы приложений – предоставляют сети прикладные сервисы.
- Игровые серверы - служат для одновременной игры нескольких пользователей в единой игровой ситуации. Некоторые игры имеют сервер в основной поставке и позволяют запускать его в невыделенном режиме (то есть позволяют играть на машине, на которой запущен сервер)
- Прочие серверы:
 - Принт-серверы позволяют пользователям сети совместно использовать общий принтер.
 - Факс-сервер позволяет пользователям сети отправлять факсимильные сообщения.

Серверные решения – ОС и/или пакеты программ, оптимизированные под выполнение компьютером функций сервера и/или содержание в своем составе комплект программ для реализации типичного набора серверов (пр.: Unix-системы, изначально предназначенные для реализации серверной инфраструктуры).

Также необходимо выделить пакеты серверов и сопутствующих программ (например комплект веб-сервер/PHP/MySQL для быстрого развертывания хостинга) для установки под Windows (для Unix свойственна модульная или «пакетная» установка каждого компонента, поэтому такие решения редки, но они существуют. Наиболее известное — LAMP).

В интегрированных серверных решениях установка всех компонентов выполняется единовременно, все компоненты в той или иной мере тесно интегрированы и предварительно настроены друг на друга. Однако в этом случае замена одного из серверов или вторичных приложений (если их возможности не удовлетворяют потребностям) может представлять проблему.

Серверные решения служат для упрощения организации базовой ИТ-инфраструктуры компаний, то есть для оперативного построения полноценной сети в компании, в том числе и «с нуля». Компоновка отдельных серверных приложений в решение подразумевает, что решение предназначено для

выполнения большинства типичных задач; при этом значительно снижается сложность развёртывания и общая стоимость владения ИТ-инфраструктурой, построенной на таких решениях.

Клиент-сервер

Это вычислительная или сетевая архитектура (фактически ПО), в которой задания или сетевая нагрузка распределены между поставщиками услуг (серверы) и заказчиками услуг (клиентами).

Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов (но м.б. расположены и на одном компьютере).

Программы-серверы ожидают от клиентских программ запросы и предоставляют им ресурсы в виде данных (пр.: загрузка файлов через HTTP, FTP, Bit Torrent, потоковые мультимедиа, а также – работа с БД) или в виде сервисных функций (пр.: работа с эл. почтой, общение в мессенджерах, просмотр web-страниц).

В дополнение к клиент-серверной модели распределенные вычислительные приложения часто используют peer-to-peer архитектуру (одноранговая, децентрализованная или пиринговая сеть – оверлейная комп. сеть, основанная на равноправии участников; часто в ней отсутствуют выделенные серверы, а каждый узел (peer) как является клиентом, так и выполняет функции сервера. В отличие от архитектуры клиент-сервера, такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов. Участниками сети являются все узлы.

Преимущества peer-to-peer:

- нагрузка и объем данных распределяются между несколькими компьютерами
- если один из них недоступен, общедоступные файлы разделяются между другими компьютерами

Преимущества клиент-сервер:

- отсутствие дублирования кода программы-сервера программами-клиентами
- ниже требования к компьютерам-клиентам
- сервер, как правило, защищен лучше большинства клиентов
- проще сформировать иерархию доступа

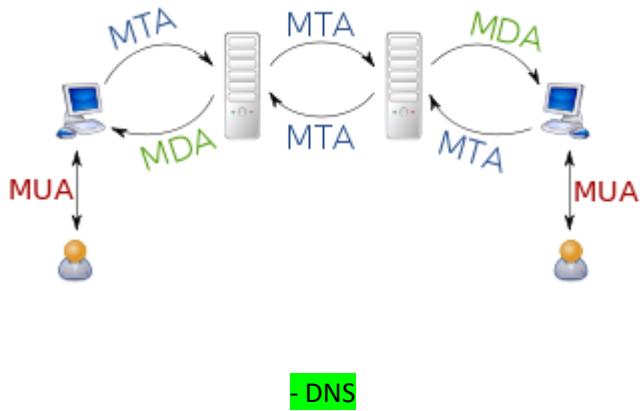
Недостатки клиент-сервер:

- неработоспособность сервера – проблема
- поддержка работы – нужен сис. админ
- дорогое железо, высокие требования к оборудованию, которые растут по мере прогнозируемого роста нагрузки

Многоуровневая архитектура «клиент — сервер» — разновидность архитектуры «клиент — сервер», в которой функция обработки данных вынесена на несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

Принцип работы эл. почты

Наиболее распространенные почтовые сервера: [gmail](#), [Sendmail](#), [Exim](#), [Postfix](#).



Domain Name System – система доменных имен – компьютерная распределенная система для получения информации о доменах.

Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получении информации о маршрутизации почты и/или обслуживающих узлах для протоколов в домене (SRV-запись).

Распределенная БД DNS поддерживается с помощью иерархии DNS-серверов, взаимодействующих по определенному протоколу.

Основа DNS – представление о иерархической структуре доменных имен и зонах. Каждый сервер, отвечающий за имя, может передать ответственность за дальнейшую часть домена другому серверу (с административной т.з. – другой организации или человеку), что позволяет возложить ответственность за актуальность информации на серверы различных организаций (людей), отвечающих только за «свою» часть доменного имени.

DNS Security Extensions (DNSSEC) – средства проверки целостности передаваемых данных.

DANE – набор спецификаций IETF, обеспечивающий аутентификацию объектов адресации (сертификатов, обеспечивающих безопасное и защищенное соединение транспортного и прикладного уровней) и предоставляемых сервисов с помощью DNS.

Ключевые характеристики DNS:

- Распределенность администрирования – ответственность за разные части иерархической структуры несут разные люди и организации.
- Распределенность хранения информации – каждый узел сети обязан хранить только те данные, которые входят в его зону ответственности, и (возможно) адреса корневых DNS-серверов.
- Кэширование информации – узел может хранить некоторое количество данных не из своей зоны ответственности для уменьшения нагрузки на сеть.
- Иерархическая структура – все узлы объединены в дерево, каждый узел может или самостоятельно определять работу нижестоящих узлов, или делегировать их другим узлам.
- Резервирование – за хранение и обслуживание своих узлов (зон) отвечают (обычно) несколько серверов, разделенные как физически, так и логически, что обеспечивает сохранность данных и продолжение работы даже в случае сбоя в одном из узлов.

Дополнительные возможности:

- поддержка динамических обновлений
- защита данных (DNSSEC) и транзакций (TSIG)
- поддержка различных типов информации

Терминология:

- Домен – узел в дереве имен вместе со всеми подчиненными ему узлами, т.е. именованная ветвь или поддерево в дереве имен. Его структура отражает порядок следования узлов в иерархии,

читая слева направо от младших доменов до доменов высшего уровня: вверху – корневой домен (имеет идентификатор «.», обслуживается корневыми серверами DNS), ниже – домены первого уровня (доменные зоны), затем – второго уровня и т.д. DNS позволяет не указывать точку корневого домена (в конце).

- Поддомен – подчиненный домен, является частью домена более высокого уровня.
- Ресурсная запись – единица хранения и передачи информации в DNS. Имеет имя (т.е. привязана к определенному доменному имени), тип и поле данных, формат и содержание которого зависят от типа.
- Зона – часть дерева доменных имен (включая ресурсные записи), размещаемая как единое целое на некотором сервере доменных имен (DNS-сервере), чаще – одновременно на нескольких серверах. Цель отделения – передача ответственности за соответствующий домен другому лицу или организации (делегирование). Являясь связной частью дерева, зона тоже является деревом с дочерними зонами.
- Делегирование – операция передачи ответственности за часть дерева доменных имен отдельному лицу или организации. Технически это выделение части дерева в отдельную зону и ее размещение на DNS-сервере под управлением лица или организации. При этом в родительскую зону включаются «склеивающие» ресурсные записи (NS и A), содержащие указатели на DNS-сервера дочерней записи.
- DNS-сервер – специализированное ПО для обслуживания DNS, а т.ж. компьютер, на котором оно выполняется. М.б. ответственным за некоторые зоны и/или может перенаправлять запросы вышестоящим серверам.
- DNS-клиент – специализированная библиотека (или программа) для работы с DNS (DNS-сервер может выступать в этой роли).
- Авторитетность – признак размещения зоны на DNS-сервере. Ответы DNS-сервера м.б. двух типов: авторитетные (сервер заявляет, что сам отвечает за зону) и неавторитетные (сервер обрабатывает запрос и возвращает ответ других серверов). Иногда вместо передачи запроса дальше DNS-сервер может вернуть уже известное ему значение (режим кэширования).
- DNS-запрос – DNS query – запрос от клиента (или сервера) к серверу. М.б. рекурсивным или не рекурсивным.

Принцип работы DNS:

Имя не тождественно IP-адресу: один IP-адрес может иметь несколько имен, что позволяет поддерживать на одном компьютере несколько веб-сайтов (виртуальный хостинг). Но и имя может иметь множество IP-адресов, что позволяет создавать балансировку нагрузки.

Для повышения устойчивости системы используется множество серверов, содержащих идентичную информацию, а в протоколе есть средства, позволяющие поддерживать синхронность информации, расположенной на разных серверах. Существует 13 корневых серверов, их адреса практически не изменяются.

Протокол DNS использует для работы TCP- или UDP-порт 53 для ответов на запросы. Традиционно запросы и ответы отправляются в виде одной UDP-датаграммы. TCP используется, когда размер данных ответа превышает 512 байт, и для AXFR-запросов.

Рекурсия:

В DNS это алгоритм поведения DNS-сервера: выполнить от имени клиента полный поиск нужной информации по всей системе DNS, при необходимости обращаясь к другим DNS-серверам.

Рекурсивный DNS-запрос требует полного поиска, нерекурсивный (итеративный) – нет.

Сам DNS-сервер м.б. рекурсивным (умеющим выполнять полный поиск) и нерекурсивным. Некоторые программы DNS-серверов, например, BIND, можно сконфигурировать так, чтобы запросы одних клиентов выполнялись рекурсивно, а запросы других — нерекурсивно.

При нерекурсивном запросе, неумении или запрете сервера выполнять рекурсивные запросы, он либо возвращает данные о зоне, за которую ответственен, либо ошибку.

Нерекурсивный сервер, выдающий информацию о серверах с большим кол-вом информации, м.б. использован для DoS-атак.

Как работает:

Предположим, мы набрали в браузере адрес ru.wikipedia.org. Браузер ищет соответствие этого адреса IP-адресу в файле hosts. Если файл не содержит соответствия, то далее браузер спрашивает у сервера DNS: «какой IP-адрес у ru.wikipedia.org»? Однако сервер DNS может ничего не знать не только о запрошенном имени, но и даже обо всём домене wikipedia.org. В этом случае сервер обращается к корневому серверу — например, 198.41.0.4. Этот сервер сообщает — «У меня нет информации о данном адресе, но я знаю, что 204.74.112.1 является ответственным за зону org.» Тогда сервер DNS направляет свой запрос к 204.74.112.1, но тот отвечает «У меня нет информации о данном сервере, но я знаю, что 207.142.131.234 является ответственным за зону wikipedia.org.» Наконец, тот же запрос отправляется к третьему DNS-серверу и получает ответ — IP-адрес, который и передаётся клиенту — браузеру.

В данном случае при разрешении имени, то есть в процессе поиска IP по имени:

- браузер отправил известному ему DNS-серверу рекурсивный запрос — в ответ на такой тип запроса сервер обязан вернуть «готовый результат», то есть IP-адрес, либо пустой ответ и код ошибки NXDOMAIN;
- DNS-сервер, получивший запрос от браузера, последовательно отправлял нерекурсивные запросы, на которые получал от других DNS-серверов ответы, пока не получил ответ от сервера, ответственного за запрошенную зону;
- остальные упоминавшиеся DNS-серверы обрабатывали запросы нерекурсивно (и, скорее всего, не стали бы обрабатывать запросы рекурсивно, даже если бы такое требование стояло в запросе).

DNS-сервер браузера кэширует результат и в следующий раз не производятся опросы прочих серверов.

Обратный DNS-запрос:

С записью DNS могут быть сопоставлены различные данные, в том числе и какое-либо символьное имя. Существует специальный домен in-addr.arpa, записи в котором используются для преобразования IP-адресов в символьные имена. Например, для получения DNS-имени для адреса 11.22.33.44 можно запросить у DNS-сервера запись 44.33.22.11.in-addr.arpa, и тот вернёт соответствующее символьное имя. Обратный порядок записи частей IP-адреса объясняется тем, что в IP-адресах старшие биты расположены в начале, а в символьных DNS-именах старшие (находящиеся ближе к корню) части расположены в конце.

Записи DNS, или ресурсные записи (resource records, RR), — единицы хранения и передачи информации в DNS. Каждая ресурсная запись состоит из следующих полей:

- имя (NAME) — доменное имя, к которому привязана или которому «принадлежит» данная ресурсная запись,
- тип (TYPE) ресурсной записи — определяет формат и назначение данной ресурсной записи,
- класс (CLASS) ресурсной записи; теоретически считается, что DNS может использоваться не только с TCP/IP, но и с другими типами сетей, код в поле класс определяет тип сети,
- TTL (Time To Live) — допустимое время хранения данной ресурсной записи в кэше неответственного DNS-сервера,
- длина поля данных (RDLEN),
- поле данных (RDATA), формат и содержание которого зависит от типа записи.

Наиболее важные типы DNS-записей:

- Запись A (address record) или запись адреса связывает имя хоста с адресом протокола IPv4. Например, запрос A-записи на имя referrals.icann.org вернёт его IPv4-адрес — 192.0.34.164.
- Запись AAAA (IPv6 address record) связывает имя хоста с адресом протокола IPv6. Например, запрос AAAA-записи на имя K.ROOT-SERVERS.NET вернёт его IPv6-адрес — 2001:7fd::1.
- Запись CNAME (canonical name record) или каноническая запись имени (псевдоним) используется для перенаправления на другое имя.
- Запись MX (mail exchange) или почтовый обменник указывает сервер(ы) обмена почтой для данного домена.
- Запись NS (name server) указывает на DNS-сервер для данного домена.
- Запись PTR (pointer) обратная DNS-запись или запись указателя связывает IP-адрес хоста с его каноническим именем. Запрос в домене in-addr.arpa на IP-адрес хоста в reverse-форме вернёт имя (FQDN) данного хоста. Например, для IP-адреса 192.0.34.164 запрос записи PTR 164.34.0.192.in-addr.arpa вернёт его каноническое имя referrals.icann.org. В целях уменьшения объёма спама многие серверы-получатели электронной почты могут проверять наличие PTR-записи для хоста, с которого происходит отправка. В этом случае PTR-запись для IP-адреса должна соответствовать имени отправляющего почтового сервера, которым он представляется в процессе SMTP-сессии.
- Запись SOA (Start of Authority) или начальная запись зоны указывает, на каком сервере хранится эталонная информация о данном домене, содержит контактную информацию лица, ответственного за данную зону, тайминги (параметры времени) кеширования зонной информации и взаимодействия DNS-серверов.
- SRV-запись (server selection) указывает на серверы для сервисов, используется, в частности, для Jabber (XMPP) и Active Directory.

Зарезервированные доменные имена – [здесь](#). Можно использовать в кач-ве примеров в документации.

Доменное имя может состоять только из ограниченного набора ASCII-символов, позволяя набрать адрес домена независимо от языка пользователя. ICANN утвердил основанную на Punycode систему IDNA, преобразующую любую строку в кодировке Unicode в допустимый DNS набор символов.

Серверы имен:

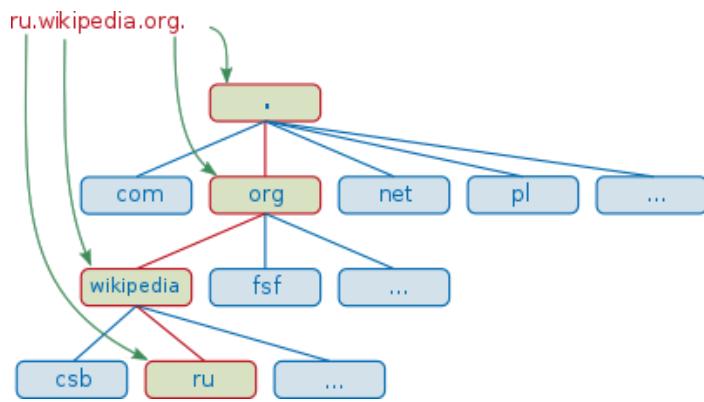
- [BIND](#) (Berkeley Internet Name Domain)
- [djbdns](#) ([Daniel J. Bernstein's DNS](#))
- [Dnsmasq](#)
- [MaraDNS](#)
- [NSD](#) (Name Server Daemon)
- [PowerDNS](#)
- [OpenDNS](#)
- [Microsoft DNS Server](#) (в серверных версиях операционных систем [Windows NT](#))
- [MyDNS](#)

- HTTP

- браузеры

- доменные имена

Пример структуры доменного имени



- архитектура сети и место бэка в ней

2. Знать особенности Python, где и как применяется в бэке, почему именно он, плюсы и минусы

4. Редактор IDE (PyCharm, VS Code, Visual Studio, IntelliJ Idea):

-какие есть

- почему PyCharm, его + и -

5. Терминал (PowerShell, Bash, Zsh) - то же, что и про редактор

6. Фреймворк (Django): плюсы и минусы, на что способен.

7. Базы данных:

- реляционные (PostgreSQL, MySQL, MS SQL, Azure Cloud SQL)

- нереляционные (NoSQL)(MongoDB, Redis, Cassandra, AWS, Firebase)

8. API (RESTful Api, Swagger)

Программный интерфейс приложения – это описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой.

Обычно входит в описание к-л интернет-протокола (пр.: RFC – Recall For Comments, заявка, содержит спецификации и стандарты, широко применяемые во всемирной сети), программного класса (фреймворка) или стандарта вызова функций ОС.

Часто реализуется отдельной программной библиотекой или сервисом ОС.

Облегчает процесс написания приложений, так как содержит набор действий, исключающих необходимость углубленного знания ряда областей. Так, например, разработчику не нужно знать операций, которые происходят в глубинах файловой системы, для того чтобы копировать файл, так как API предоставит функцию для данного действия.

Если рассмотреть программу (библиотеку, модуль) как черный ящик, API поможет им управлять.

Программный компоненты иерархично взаимодействуют посредством API (высокоуровневые компоненты используют API низкоуровневых и т.д. по нисходящей).

По такому принципу построены протоколы передачи данных по интернету. Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью нижестоящего уровня и предоставляет функциональность вышестоящему.

Понятие протокола близко по смыслу к понятию API: оба являются абстракцией функциональности, только первое связано с передачей данных, а второе – со взаимодействием приложений.

API библиотеки функций и классов включают в себя описание сигнатур и семантики функций.

- Сигнатура функции – часть общего объявления функции, позволяющая средствам трансляции идентифицировать функцию среди других.

Т.к. в разных языках разные представления о сигнатуре функций, есть возможность пользоваться перезагрузкой функции, т.е. использовать функции с одинаковыми названиями.

Также различают сигнатуру вызова (составляется по синтаксической конструкции вызова функции с учетом ее области видимости, ее имени, последовательности фактических типов аргументов в вызове и типа результата) и сигнатуру реализации функции (в ней участвуют некоторые элементы из синтаксической конструкции объявления функции: спецификатор области видимости функции, ее имя и последовательность формальных типов аргументов).

- Семантика функции – описание того, что эта функция делает. Включает в себя описание того, что является результатом вычисления функции, как и от чего этот результат зависит. Он может зависеть от аргументов, а иногда и от состояния. Логика этих зависимостей и изменений относится к семантике функции.

У каждой ОС имеют API, которое позволяет создавать приложения для данной ОС. Главный API ОС – множество системных вызовов (обращений прикладной программы к ядру ОС для выполнения к-л операции).

Единые стандарты API внутри одной ОС гарантируют правильную и схожую работу программ, написанных в рамках этого API.

Однако различия API разных ОС приводят к затруднению переноса приложений между платформами. Возможные пути решения:

- написание «промежуточных» API (API графических интерфейсов wxWidgets, GTK и т. п.)
- написание библиотек, которые отображают системные вызовы одной ОС в системные вызовы другой (Wine, cygwin и т. п.)
- введение стандартов кодирования в языках программирования (пр.: стандартная библиотека языка C)
- написание интерпретируемых языков, реализуемых на разных платформах (sh, Python, Perl, PHP, Tcl, JavaScript, Ruby и т. д.)

Также в распоряжении программиста часто находится несколько различных API, позволяющих добиться одного и того же результата.

Основными сложностями существующих многоуровневых систем API, таким образом, являются:

- Сложность портирования программного кода с одной системы API на другую (например, при смене ОС);
- Потеря функциональности при переходе с более низкого уровня на более высокий. Грубо говоря, каждый «слой» API создаётся для облегчения выполнения некоторого стандартного набора

операций. Но при этом реально затрудняется либо становится принципиально невозможным выполнение некоторых других операций, которые предоставляет более низкий уровень API.

Наиболее известные API:

- ОС: [Amiga ROM Kernel](#), [Cocoa](#), [Linux Kernel API](#), [OS/2 API](#), [POSIX](#), [Windows API](#)
- Графических интерфейсов: [DirectDraw](#)/[Direct3D](#) (часть [DirectX](#)), [GDI](#), [GDI+](#), [GTK](#), [SFML](#), [Motif](#), [OpenGL](#), [OpenVG](#), [Qt](#), [SDL](#), [Vulkan](#), [Tk](#), [wxWidgets](#), [X11](#), [Zune](#)
- Звуковых интерфейсов: [DirectMusic](#)/[DirectSound](#) (часть [DirectX](#)), [OpenAL](#)
- Аутентификационных систем: [BioAPI](#), [PAM](#)

Web API – используется в веб-разработке, содержит определенный набор HTTP-запросов, а т.ж. определение структуры HTTP-ответов, для выражения которых используют XML – или JSON-формат.

Сейчас перешли от SOAP к REST типу коммуникации.

Веб-интерфейсы, обеспечивающие сочетание нескольких сервисов в новых приложениях, известны как гибридные.

Структуры данных

Статический массив

Динамический массив

Ассоциативный массив (HASHTABLE)

9. Аутентификация (OAuth 2.0, JWT):

- Провайдеры (Auth0, Firebase)

10. Паттерны проектирования (чит. кн. “Банда 4x”):

- порождающие

- структурные

- поведенческие

11. Тесты (+изучить фреймворки для написания этих тестов):

- Unit

- интеграционные

12. Доп. инструменты разработки

- менеджеры пакетов (NuGet, npm, yarn)

- системы контроля версий (git, GitHub, TFS, BitBucket)

Также:

- умение декомпозировать предметную область (понять уже написанный код, легко в нем ориентироваться)
- знание вэб-сервисов, умение их создавать
- сетевые протоколы (как открыть порт, к нему приконнектиться, открыть приложение вызовов)
- ui фреймфорки

Unix-подобная ОС

Это свободные/открытые ОС, созданные по подобию Unix (пр.: Linux, FreeBSD).

Операционные системы семейства Unix характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой, взаимодействие осуществляется через единую файловую систему, а для работы с утилитами используется командная оболочка.

Межпроцессное взаимодействие (англ. inter-process communication, IPC)

Это обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

Из механизмов, предоставляемых ОС и используемых для IPC, можно выделить:

- механизмы обмена сообщениями;
- механизмы синхронизации;
- механизмы разделения памяти;
- механизмы удалённых вызовов (RPC).

Для оценки производительности различных механизмов IPC используют следующие параметры:

- пропускная способность (количество сообщений в единицу времени, которое ядро ОС или процесс способно обработать);
- задержки (время между отправкой сообщения одним потоком и его получением другим потоком).

Именованный канал

В программировании именованный канал или именованный конвейер (англ. named pipe) — один из методов межпроцессного взаимодействия, расширение понятия конвейера в Unix и подобных ОС.

Именованный канал позволяет различным процессам обмениваться данными, даже если программы, выполняющиеся в этих процессах, изначально не были написаны для взаимодействия с другими программами.

Это понятие также существует и в Microsoft Windows, хотя там его семантика существенно отличается. Традиционный канал — «безымянен», потому что существует анонимно и только во время выполнения процесса.

Именованный канал — существует в системе и после завершения процесса. Он должен быть «отсоединен» или удалён, когда уже не используется. Процессы обычно подсоединяются к каналу для осуществления взаимодействия между ними.

Сокет (программный интерфейс)

Это название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. [Сокет](#) — абстрактный объект, представляющий конечную точку соединения.

Следует различать клиентские и серверные сокеты. Клиентские сокеты грубо можно сравнить с конечными аппаратами телефонной сети, а серверные — с коммутаторами. Клиентское приложение (пр.: браузер) использует только клиентские сокеты, а серверное (пр.: веб-сервер, которому браузер посыпает запросы) — как клиентские, так и серверные сокеты.

Майнфрейм (также [майнфрейм](#), от англ. mainframe) — большой универсальный высокопроизводительный отказоустойчивый сервер со значительными ресурсами ввода-вывода, большим объёмом оперативной и внешней памяти, предназначенный для использования в критически важных системах (англ. mission-critical) с интенсивной пакетной и оперативной транзакционной обработкой.

Файловый сервер

Выделенный сервер, предназначенный для файловых операций ввода-вывода и хранящий файлы любого типа. Как правило, обладает большим объемом дискового пространства, реализованным в формате RAID-массива для обеспечения бесперебойной работы и повышенной скорости записи и чтения данных. Есть также [файл-серверные приложения](#).

Двоичный интерфейс приложений

Он же [Application Binary Interface](#) (ABI) — набор соглашений для доступа приложения к ОС и другим низкоуровневым сервисам, спроектированный для переносимости исполняемого кода между машинами, имеющими совместимые ABI.

В отличие от API, который регламентирует совместимость на уровне исходного кода, ABI можно рассматривать как совокупность правил, позволяющих компоновщику объединять откомпилированные модули компонента без перекомпиляции всего кода, в то же время определяя двоичный интерфейс.

Двоичный интерфейс регламентирует:

- использование регистров процессора
- состав и формат системных вызовов и вызова одного модуля другим
- формат передачи аргументов и возвращаемого значения при вызове функции

Двоичный интерфейс приложений описывает функциональность, предоставляемую рядом ОС и архитектурой набора команд (без привилегированных команд).

Если интерфейс программирования разных платформ совпадает (API), код для этих платформ можно компилировать без изменений.

Если совпадают и API, и ABI, исполняемые файлы можно переносить на эти платформы без изменений.

Если API или ABI платформ различаются, код требует изменений и повторной компиляции.

API не обеспечивает совместимости среды исполнения программы — это задача двоичного интерфейса.

Есть также бинарный интерфейс встраиваемых приложений (Embedded ABI, EABI) — набор соглашений для использования во встраиваемом ПО, описывающий:

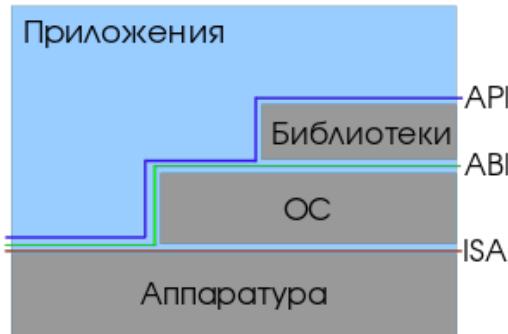
- [форматы файлов](#)

- [типы данных](#)
- способы использования регистров
- организацию [стека](#)
- соглашение о вызове функции.

Если объектный файл был создан компилятором, поддерживающим EABI, становится возможной компоновка этого объектного файла любым компоновщиком, поддерживающим тот же EABI.

Основное отличие EABI от ABI в ОС общего назначения заключается в том, что в коде приложения допускаются привилегированные команды, а динамическое связывание (компоновка) не требуется (а иногда и полностью запрещена), а также, в целях экономии памяти, используется более компактная организация стека.

Уровни и интерфейсы между ними. API, ABI и архитектура набора команд (ISA)



IP-адрес – [Internet Protocol](#) – уникальный числовой идентификатор устройства в компьютерной сети, работающий по протоколу TCP/IP. Состоит из двух частей: номера сети и номера узла.

Хост – [host](#) – любое устройство, предоставляющее сервисы формата «клиент-сервер» в режиме сервера по каким-либо интерфейсам и уникально определенное на этих интерфейсах. Глобально – любой компьютер, подключенный к локальной или глобальной сети.

Чаще всего, под «хостом» без дополнительных комментариев подразумевается хост протокола TCP/IP, то есть сетевой интерфейс устройства, подключённого к IP-сети. Как и всякий другой хост, этот имеет уникальное определение в среде сервисов TCP/IP (IP-адрес). С хостом протокола TCP/IP может быть также связана необязательная текстовая характеристика — доменное имя.

Служебная запись (SRV-запись) – стандарт в DNS, определяющий местоположение, т.е. имя хоста и номер порта серверов для определенных служб.

Некоторые Интернет-протоколы, такие как SIP и XMPP, часто требуют поддержки SRV-записей.

SRV-запись имеет такой формат:

```
_service._proto.name TTL class SRV priority weight port target
```

- service: символьное имя сервиса.
- proto: транспортный протокол, используемый сервисом, как правило TCP или UDP.
- name: доменное имя, для которого эта запись действует.
- TTL: стандарт DNS, время жизни.
- class: стандарт DNS, поле класса (это всегда IN).
- priority: приоритет целевого хоста, более низкое значение означает более предпочтительный.
- weight: относительный вес для записей с одинаковым приоритетом.

- port: Порт TCP или UDP, на котором работает сервис.
- target: канонические имя машины, предоставляющей сервис.

Распределенная БД (параллельная) – [Distributed Database](#), DDB – единая БД, составные части которой размещаются в различных узлах компьютерной сети в соответствии с к-л критерием.

Данные я-ся DDB только если они связаны в соответствии с некоторым структурным формализмом, реляционной моделью, а доступ к ним обеспечивается единым высокогорневым интерфейсом.

Могут иметь разный уровень реплицированности – от полного отсутствия дублирования информации до дублирования всей информации во всех распределенных копиях (пр.: [блокчейн](#)).

Прозрачность – распределение БД по множеству узлов невидимо для пользователей. Полная прозрачность достигается за счет: прозрачности сети, распределения, репликации, фрагментации, доступа.

Виртуальный хостинг - [shared hosting](#) – при нем множество веб-сайтов расположено на одном веб-сервере, каждый в своем разделе, по все используют одно ПО.

Существует два основных метода реализации доступа к веб-сайтам:

- по имени shared IP hosting), когда все веб-сайты используют один общий IP-адрес. Согласно протоколу HTTP/1.1, веб-браузер при запросе к веб-серверу указывает доменное имя веб-сайта в поле Host заголовка текущего запроса, и веб-сервер использует его для правильного выполнения запроса, а также копирует это имя в ячейку [HTTP_HOST] суперглобального массива \$_SERVER.
- по IP-адресу (dedicated IP hosting), при котором у каждого веб-сайта есть собственный IP-адрес, а веб-сервер имеет несколько физических или виртуальных сетевых интерфейсов.

Сертификат открытого ключа (сертификат электронной подписи, сертификат ключа подписи, сертификат ключа проверки электронной подписи) – электронный или бумажный документ, содержащий открытый ключ, информацию о владельце ключа, области применения ключа, подписанный выдавшим его Удостоверяющим центром и подтверждающий принадлежность [открытого ключа](#) владельцу.

Протокол передачи данных – [набор](#) определённых правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

TCP/IP – [набор](#) протоколов передачи данных, получивший название от двух принадлежащих ему протоколов: TCP (Transmission Control Protocol) и IP (Internet Protocol)

HTTP (Hyper Text Transfer Protocol) – это [протокол](#) передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц между компьютерами, подключёнными к одной сети.

FTP (File Transfer Protocol) – это [протокол](#) передачи файлов со специального файлового сервера на компьютер пользователя. FTP даёт возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удалённым компьютером, пользователь может скопировать файл с удалённого компьютера на свой или скопировать файл со своего компьютера на удалённый.

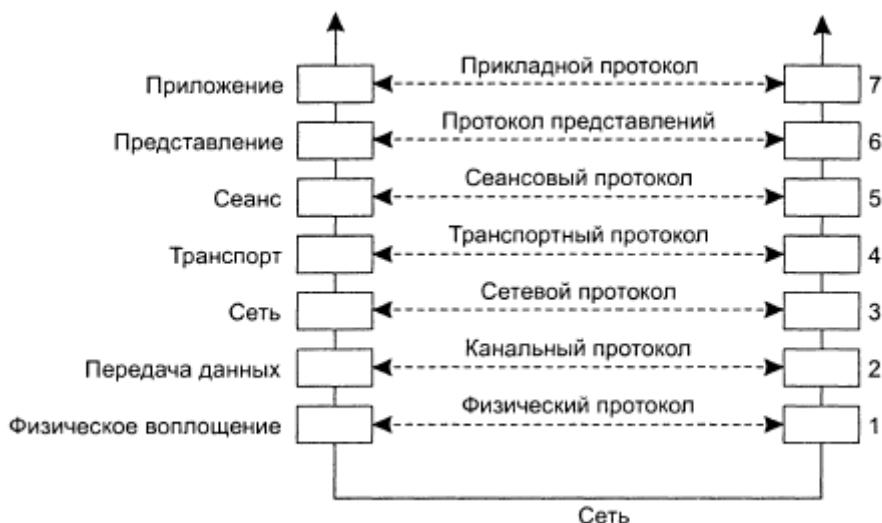
POP3 (Post Office Protocol) – это стандартный [протокол](#) почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.

SMTP (Simple Mail Transfer Protocol) — [протокол](#), который задаёт набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приёме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.

TELNET — это [протокол](#) удалённого доступа. TELNET даёт возможность абоненту работать на любой ЭВМ, находящейся с ним в одной сети, как на своей собственной, то есть запускать программы, менять режим работы и так далее. На практике возможности ограничиваются тем уровнем доступа, который задан администратором удалённой машины.

Модель OSI — 7-уровневая логическая модель работы сети. Реализуется группой протоколов и правил связи, организованных в несколько уровней:

- на физическом уровне определяются физические (механические, электрические, оптические) характеристики линий связи;
- на канальном уровне определяются правила использования физического уровня узлами сети;
- сетевой уровень отвечает за адресацию и доставку сообщений;
- транспортный уровень контролирует очерёдность прохождения компонентов сообщения;
- сеансовый уровень координирует связь между двумя прикладными программами, работающими на разных рабочих станциях;
- уровень представления служит для преобразования данных из внутреннего формата компьютера в формат передачи;
- прикладной уровень является пограничным между прикладной программой и другими уровнями, обеспечивая удобный интерфейс связи для сетевых программ пользователя.



Модель — стек протоколов TCP/IP — содержит 4 уровня:

- канальный уровень (link layer),
- сетевой уровень (Internet layer),
- транспортный уровень (transport layer),
- прикладной уровень (application layer).



Демон (*daemon*) — компьютерная программа в UNIX-подобных системах, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем.

Демоны обычно запускаются во время загрузки системы. Типичные задачи демонов: серверы сетевых протоколов (HTTP, FTP, электронная почта и др.), управление оборудованием, поддержка очередей печати, управление выполнением заданий по расписанию и т. д. В техническом смысле демоном считается процесс, который не имеет управляющего терминала. Традиционно названия демон-процессов заканчиваются на букву d.

В системах Windows аналогичный класс программ называется службой (Services).

TCP/IP — сетевая [модель](#) четырехуровневой передачи цифровых данных от источника к получателю.

Каждый уровень описан протоколом передачи. На стеке протоколов передачи данных базируется Интернет.

[TCP](#) — Transmission Control Protocol — протокол управления передачей данных (пакеты – сегменты).

[IP](#) — Internet Protocol — масштабируемый протокол сетевого уровня, который объединил отдельные компьютерные сети в глобальную сеть Интернет.

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься.

Уровни:

- [Прикладной уровень](#) (Application Layer) - обеспечивает обмен данными между процессами для приложений (пр.: [HTTP](#), [RTSP](#), [FTP](#), [DNS](#));

На нем работает большинство сетевых приложений. Они имеют собственные протоколы обмена данными (пр.: [интернет браузер](#) для протокола [HTTP](#), [ftp-клиент](#) для протокола [FTP](#) (передача файлов), почтовая программа для протокола [SMTP](#) ([электронная почта](#)), [SSH](#) (безопасное соединение с удалённой машиной), [DNS](#) (преобразование символьных имён в [IP-адреса](#)) и др.)

В массе своей эти протоколы работают поверх [TCP](#) или [UDP](#) и привязаны к определённому [порту](#), например:

- [HTTP](#) на TCP-порт 80 или 8080,
- [FTP](#) на TCP-порт 20 (для передачи данных) и 21 (для управляющих команд),
- [SSH](#) на TCP-порт 22,
- запросы [DNS](#) на порт UDP (реже TCP) 53,
- обновление маршрутов по протоколу [RIP](#) на UDP-порт 520.

К этому уровню относятся: [Echo](#), [Finger](#), [Gopher](#), [HTTP](#), [HTTPS](#), [IMAP](#), [IMAPS](#), [IRC](#), [NNTP](#), [NTP](#), [POP3](#),

[POPS](#), [QOTD](#), [RTSP](#), [SNMP](#), [SSH](#), [Telnet](#), [XDMCP](#).

- [Транспортный уровень](#) (Transport Layer) - обрабатывающий связь между хостами (пр.: [TCP](#), [UDP](#), [SCTP](#), [DCCP](#); [RIP](#), протоколы маршрутизации, подобные [OSPF](#), что работают поверх [IP](#), являются частью сетевого уровня);

Могут решать проблему негарантированной доставки сообщений, а т.ж. гарантировать правильную последовательность прихода данных. В стеке TCP/IP определяют, для какого именно приложения эти данные.

Протоколы автоматической маршрутизации, логически представленные на этом уровне (поскольку работают поверх IP), на самом деле являются частью протоколов сетевого уровня; например [OSPF](#) (IP идентификатор 89).

[TCP](#) (IP идентификатор 6) — «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный [поток данных](#), дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности. В этом его главное отличие от [UDP](#).

[UDP](#) (IP идентификатор 17) протокол передачи [датаграмм](#) без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов. В приложениях, требующих гарантированной передачи данных, используется протокол [TCP](#).

И [TCP](#), и [UDP](#) используют для определения протокола верхнего уровня число, называемое [портом](#).

- [Межсетевой уровень](#) (Сетевой уровень) (Internet Layer) - обеспечивающий межсетевое взаимодействие между независимыми сетями (пр.: для TCP/IP это [IP](#) (вспомогательные протоколы, вроде [ICMP](#) и [IGMP](#), работают поверх IP, но тоже относятся к сетевому уровню; протокол [ARP](#) является самостоятельным вспомогательным протоколом, работающим поверх канального уровня);

Разработан для передачи данных из любой сети в любую сеть, независимо от протоколов нижнего уровня, а также может запрашивать данные от удалённой стороны, например в протоколе [ICMP](#) (используется для передачи диагностической информации [IP](#)-соединения) и [IGMP](#) (используется для управления [multicast](#)-потоками).

На этом уровне работают [маршрутизаторы](#), которые перенаправляют пакеты в нужную сеть путём расчёта адреса сети по [маске сети](#). Примерами такого протокола является [X.25](#) и [IPC](#) в сети [ARPANET](#).

ICMP и IGMP расположены над [IP](#) и должны попасть на следующий — транспортный — уровень, но функционально являются протоколами сетевого уровня, и поэтому их невозможно вписать в модель OSI.

Пакеты сетевого протокола [IP](#) могут содержать код, указывающий, какой именно протокол следующего уровня нужно использовать, чтобы извлечь данные из пакета. Это число — уникальный [IP-номер протокола](#). ICMP и IGMP имеют номера, соответственно, 1 и 2.

К этому уровню относятся: [DVMRP](#), [ICMP](#), [IGMP](#), [MARS](#), [PIM](#), [RIP](#), [RIP2](#), [RSVP](#)

- [Канальный уровень](#) (Network Access Layer, Link Layer) - содержащий методы связи для данных, которые остаются в пределах одного сегмента сети (пр.: [Ethernet](#), [IEEE 802.11 WLAN](#), [SLIP](#), [Token Ring](#), [ATM](#) и [MPLS](#), физическая среда и принципы кодирования информации, [T1](#), [E1](#)).

Описывает способ кодирования данных для передачи [пакета данных](#) на физическом уровне (то есть специальные последовательности бит, определяющих начало и конец пакета данных, а также обеспечивающие помехоустойчивость). [Ethernet](#), например, в полях [заголовка пакета](#) содержит указание того, какой машине или машинам в сети предназначен этот пакет.

Примеры протоколов канального уровня — [Ethernet](#), [IEEE 802.11 WLAN](#), [SLIP](#), [Token Ring](#), [ATM](#) и [MPLS](#).

[PPP](#) не совсем вписывается в такое определение, поэтому обычно описывается в виде пары протоколов [HDLC/SDLC](#).

[MPLS](#) занимает промежуточное положение между канальным и сетевым уровнем и, строго говоря, его нельзя отнести ни к одному из них.

Канальный уровень иногда разделяют на 2 подуровня — [LLC](#) и [MAC](#).

Кроме того, канальный уровень описывает среду передачи данных (будь то коаксиальный кабель, витая пара, оптическое волокно или радиоканал), физические характеристики такой среды и принцип передачи данных (разделение каналов, модуляцию, амплитуду сигналов, частоту сигналов, способ синхронизации передачи, время ожидания ответа и максимальное расстояние).

При проектировании стека протоколов на канальном уровне рассматривают помехоустойчивое кодирование — позволяющие обнаруживать и исправлять ошибки в данных вследствие воздействия шумов и помех на канал связи.

Распределение протоколов по уровням модели OSI

TCP/IP		OSI
7	Прикладной	напр., HTTP , SMTP , SNMP , FTP , Telnet , SSH , SCP , SMB , NFS , RTSP , BGP
6	Прикладной	напр., XDR , AFP , TLS , SSL
5	Сеансовый	напр., ISO 8327 / CCITT X.225 , RPC , NetBIOS , PPTP , L2TP , ASP
4	Транспортный	напр., TCP , UDP , SCTP , SPX , ATP , DCCP , GRE
3	Сетевой	напр., IP , ICMP , IGMP , CLNP , OSPF , RIP , IPX , DDP
2	Канальный	напр., Ethernet , Token ring , HDLC , PPP , X.25 , Frame relay , ISDN , ATM , SPB , MPLS , ARP
1	Физический	напр., электрические провода , радиосвязь , волоконно-оптические провода , инфракрасное излучение

Сетевая модель OSI ([The Open System Interconnection model](#)) – сетевая модель стека (магазина) сетевых протоколов OSI/ISO, посредством которой различные сетевые устройства могут взаимодействовать друг с другом; определяет различные уровни взаимодействия систем, у каждого из которых свои функции.

Протоколы связи позволяют структуре на одном хосте взаимодействовать с соответствующей структурой того же уровня на другом хосте.

На каждом уровне N два объекта обмениваются блоками данных ([PDU](#)) с помощью протокола данного уровня на соответствующих устройствах. Каждый PDU содержит блок служебных данных ([SDU](#)), связанный с верхним или нижним протоколом.

Обработка данных двумя взаимодействующими OSI-совместимыми устройствами происходит следующим образом:

1. Передаваемые данные составляются на самом верхнем уровне передающего устройства (уровень N) в протокольный блок данных (PDU).
2. PDU передается на уровень N-1, где он становится сервисным блоком данных (SDU).
3. На уровне N-1 SDU объединяется с верхним, нижним или обоими уровнями, создавая слой N-1 PDU. Затем он передается в слой N-2.
4. Процесс продолжается до достижения самого нижнего уровня, с которого данные передаются на принимающее устройство.
5. На приемном устройстве данные передаются от самого низкого уровня к самому высокому в виде серии SDU, последовательно удаляясь из верхнего или нижнего колонтитула каждого слоя до достижения самого верхнего уровня, где принимаются последние данные.

Модель					
	Уровень (layer)	Тип данных (PDU ^[14])	Функции	Примеры	Оборудование
Host layers	7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3, WebSocket	Хосты (клиенты сети), Межсетевой экран
	6. Представления (presentation)		Представление и шифрование данных	ASCII, EBCDIC, JPEG, MIDI	
	5. Сеансовый (session)		Управление сеансом связи	RPC, PAP, L2TP, gRPC	
	4. Транспортный (transport)	Сегменты (segment) / Датаграммы (datagram)	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP, Порты	
Media^[15] layers	3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk, ICMP	Маршрутизатор, Сетевой шлюз, Межсетевой экран
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация	PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.	Сетевой мост, Коммутатор, точка доступа
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	USB, RJ («витая пара», коаксиальный, оптоволоконный), радиоканал	Концентратор, Повторитель (сетевое оборудование)

Web-сервер – [сервер](#), принимающий HTTP-запросы от клиентов, обычно веб-браузеров (обозначены URL-адресами), и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Дополнительные функции:

- автоматизация работы веб-страниц
- ведение журнала обращений пользователя к ресурсам
- аутентификация и авторизация пользователей

- поддержка динамически генерируемых страниц
- поддержка HTTPS для защищенных соединений с клиентами.

65% рынка занимает web-сервер Apache – свободный веб-сервер, наиболее часто используемый в Unix-подобных ОС.

Прочие:

- [IIS](#) от компании [Microsoft](#), распространяемый с ОС семейства [Windows](#).
- [nginx](#) – свободный веб-сервер, разрабатываемый [Игорем Сысоевым](#) с 2002 года и пользующийся большой популярностью на крупных сайтах^[2] ^[3],
- [lighttpd](#) – свободный веб-сервер.
- [Google Web Server](#) – веб-сервер разработанный компанией [Google](#).
- [Resin](#) – свободный веб-сервер приложений.
- [Cherokee](#) – свободный веб-сервер, управляемый только через web-интерфейс.
- [Rootage](#) – веб-сервер, написанный на [Java](#).
- [THTTPD](#) – простой, маленький, быстрый и безопасный веб-сервер.
- [Open Server](#) – бесплатная программа с графическим интерфейсом использует множество исключительно свободного программного комплекса.
- [H2O](#) – свободный быстрый веб-сервер, написанный на [C](#).
- [nghhttp2](#) – веб-сервер, встроенный в [Node.js](#).
- [Go HTTP](#) – веб-сервер, встроенный в [Go](#).

Клиенты:

- веб-браузер, работающий на ПК или переносном устройстве (пр.: кПК)
- программы, самостоятельно обращающиеся к веб-серверам для получения обновлений или другой информации (пр.: антивирус запрашивает обновление БД)
- мобильный телефон, получающий доступ к ресурсам веб-сервера через WAP
- др. цифровые устройства и бытовая техника.

Отличие веб-сервера от сервера приложений: веб-сервер предназначен для обслуживания статических страниц (пр.: HTML, CSS), а сервер приложений отвечает за генерацию динамического содержимого путем выполнения кода на стороне сервера (пр.: JSP, EJB и др.).

Доменное имя – [domain name](#) – символическое имя, служащее для идентификации областей, которые являются единицами административной автономии в сети Интернет, в составе вышестоящей по иерархии такой области. Каждая такая область – домен. Общее пространство имен функционирует благодаря DNS.

Доменные адреса дают возможность адресации Интернет-узлов и расположенным на них ресурсам (веб-сайтам, серверам, серверам эл. почты, другим службам) быть предоставленными в удобной для человека форме.

Структура полного доменного имени:

Непосредственное имя домена + имена всех доменов, в которые он входит, разделенные точками.

FQDN (fully qualified domain name) – полное доменное имя, т.е. не имеющее неоднозначностей в определении, включающее в себя имена всех родительских доменов иерархии DNS.

В DNS, а точнее в zone file, FQDN завершаются точкой, т.е. включают корневое имя «.», которое является безымянным (на практике опускается).

Различия между FQDN и доменным именем проявляются при именовании доменов второго, третьего и т.д. уровней. Для FQDN обязательно указать домены более высокого уровня.

В DNS-записях доменов (для перенаправления (трансляция порт-адрес – технология трансляции порт-адреса в зависимости от TCP-UDP-порта получателя), почтовых серверов и т.д.) всегда используется FQDN.

Доменная зона – совокупность доменных имен определенного уровня, входящих в конкретный домен (термин применяется в тех. сфере при настройке DNS-серверов: поддержание, делегирование и трансфер зоны).

DNS-рэзерверы – отвечают на вопросы, касающиеся любых зон.

Служба whois – позволяет узнать владельца (админа) домена.

Дроп-домен – у него истек срок регистрации, и теперь он свободен.

Виды:

- Тематические домены (gTLD). [Общие домены верхнего уровня](#) (gTLD) управляются организацией [ICANN](#).
- Интернационализированные домены (IDN). Доменные имена, которые содержат символы национальных алфавитов. [IDN](#) верхнего уровня управляются и находятся под контролем [ICANN](#).
- Национальные домены (ccTLD). [Национальные домены верхнего уровня](#) (ccTLD) делегированы соответствующим национальным регистраторам, которые устанавливают правила регистрации в них либо сами, либо согласно указаниям правительства. Управляющей организацией является [IANA](#).
- Зарезервированные доменные имена. Документ [RFC 2606](#) (Reserved Top Level DNS Names — Зарезервированные имена доменов верхнего уровня) определяет названия доменов, которые следует использовать в качестве примеров (например, в документации), а также для тестирования. Кроме [example.com](#), [example.org](#) и [example.net](#), в эту группу также входят [.test](#), [.invalid](#) и др.
- Длинные доменные имена. Размер доменного имени ограничивается по административным и техническим причинам. Обычно разрешается [регистрация доменов](#) длиной до 63 символов. В некоторых странах можно регистрировать домены длиной до 127 знаков.

Apache HTTP-сервер – [апач](#) – свободный веб-сервер.

Это кроссплатформенное ПО, поддерживает операционные системы [Linux](#), [BSD](#), [Mac OS](#), [Microsoft Windows](#), [Novell NetWare](#), [BeOS](#).

Его основные достоинства – надежность и гибкость конфигурации. Он позволяет переключать внешние модули для предоставления данных, использовать СУБД для аутентификации пользователей, модифицировать сообщения об ошибках и т.д. Поддерживает IPv4.

Архитектура:

- Ядро – включает в себя основные функциональные возможности (обработка конфигурационных файлов, протокол HTTP и система загрузки модулей). Язык – С.
- Система конфигурации – основана на текстовых конфигурационных файлах, имеет 3 уровня конфигурации:
 - К. сервера (`httpd.conf`) – директивы к. сгруппированы в 3 осн. раздела: 1) управляющие процессом Apache в целом (глобальное окружение); 2) определяющие параметры «главного» сервера, или сервера по умолчанию, который отвечает на запросы, которые не обрабатываются виртуальными хостами (определяют т.ж. установки по умолчанию для всех остальных виртуальных хостов); 3) установки для виртуальных хостов, позволяющие обрабатывать запросы Web одним единственным сервером Apache, но направлять по разным адресам IP или именам хостов.
 - К. виртуального хоста (`httpd.conf` с версии 2.2, `extra/httpd-vhosts.conf`).
 - К. уровня каталога (`.htaccess`).

Имеет собственный язык конфигурационных файлов, основанный на блоках директив. Практически все параметры ядра могут быть изменены через конфигурационные файлы, вплоть до управления MPM. Большая часть модулей имеет собственные параметры.

Часть модулей использует в своей работе конфигурационные файлы операционной системы (например [/etc/passwd](#) и [/etc/hosts](#)).

Помимо этого, параметры могут быть заданы через ключи [командной строки](#).

- **Многопроцессорные модели (MPM)**.

Для веб-сервера Apache существует множество моделей [симметричной многопроцессорности](#).

Вот основные из них:

Название	Разработчик	Поддерживаемые OS	Описание	Назначение	Статус
worker	Apache Software Foundation	Linux, FreeBSD	Гибридная многопроцессорно-многопоточная модель. Сохраняя стабильность многопроцессорных решений, она позволяет обслуживать большое число клиентов с минимальным использованием ресурсов.	Среднезагруженные веб-серверы.	Стабильный.
pre-fork	Apache Software Foundation	Linux, FreeBSD	MPM, основанная на предварительном создании отдельных процессов, не использующая механизм threads.	Большая безопасность и стабильность за счёт изоляции процессов друг от друга, сохранение совместимости со старыми библиотеками, не поддерживающими threads.	Стабильный.
perchild	Apache Software Foundation	Linux	Гибридная модель, с фиксированным количеством процессов.	Высоконагруженные серверы, возможность запуска дочерних процессов используя другое имя пользователя для повышения безопасности.	В разработке, нестабильный.
netware	Apache Software Foundation	Novell NetWare	Многопоточная модель, оптимизированная для работы в среде NetWare.	Серверы Novell NetWare	Стабильный.
winnt	Apache Software Foundation	Microsoft Windows	Многопоточная модель, созданная для операционной системы Microsoft Windows.	Серверы под управлением Windows Server.	Стабильный.
Apache-ITK	Steinar H. Gunderson	Linux, FreeBSD	MPM, основанная на модели prefork. Позволяет запуск каждого виртуального хоста под отдельными uid и gid.	Хостинговые серверы, серверы, критичные к изоляции пользователей и учёту ресурсов.	Стабильный.
peruser	Sean Gabriel Heacock	Linux, FreeBSD	Модель, созданная на базе MPM perchild. Позволяет запуск каждого виртуального хоста под отдельными uid и gid. Не использует потоки.	Обеспечение повышенной безопасности, работа с библиотеками, не поддерживающими threads.	Стабильная версия от 4 октября 2007 года, экспериментальная — от 10 сентября 2009 года.
event	Apache Software Foundation	Linux, FreeBSD	Модель использует threads и thread-safe polling основана на worker. Предназначен для одновременного обслуживания большего количества запросов путем передачи некоторой обработки в потоки слушателей, освобождая рабочие потоки для обслуживания новых запросов.	Обеспечение повышенной производительности. Не очень хорошо работает на старых платформах, в которых отсутствует хорошая многопоточность, но требование EPoll или KQueue делает это спорным.	Стабильный.

- **Система модулей**.

Apache HTTP Server поддерживает [модульность](#). Модули могут быть как включены в состав сервера в момент [компиляции](#), так и загружены динамически, через директивы конфигурационного файла.

В модулях реализуются такие вещи, как:

- Поддержка [языков программирования](#).
- Добавление функций.
- Исправление ошибок или модификация основных функций.
- Усиление [безопасности](#).

Часть веб-приложений, например панели управления [ISPmanager](#) и [VDSmanager](#) реализованы в виде модуля Apache.

- **Механизм виртуальных хостов**.

Apache имеет встроенный механизм виртуальных [хостов](#). Он позволяет полноценно обслуживать на одном [IP-адресе](#) множество [сайтов \(доменных имён\)](#), отображая для каждого из них собственное содержимое.

Для каждого виртуального хоста можно указать собственные настройки ядра и модулей, ограничить доступ ко всему сайту или отдельным файлам. Некоторые МРМ, например Apache-ITK, позволяют запускать [процесс](#) httpd для каждого виртуального хоста с отдельными идентификаторами [uid](#) и [guid](#).

Также существуют модули, позволяющие учитывать и ограничивать ресурсы [сервера](#) ([CPU](#), [RAM](#), [трафик](#)) для каждого виртуального хоста.

Функциональные возможности:

- [Интеграция с другим ПО и языками программирования](#)

Существует множество модулей, добавляющих к Apache поддержку различных [языков программирования](#) и систем разработки.

К ним относятся:

- [PHP](#) (mod_php).
- [Python](#) ([mod python](#), [mod wsgi](#)).
- [Ruby](#) (apache-ruby).
- [Perl](#) ([mod perl](#)).
- [ASP](#) (apache-asp).
- [Tcl](#) (rivet)

Кроме того, Apache поддерживает механизмы [CGI](#) и [FastCGI](#), что позволяет исполнять программы на практических всех языках программирования, в том числе [C](#), [C++](#), [Lua](#), [sh](#), [Java](#).

- [Безопасность](#)

Apache имеет различные механизмы обеспечения безопасности и разграничения доступа к данным. Основными являются:

- Ограничение доступа к определённым каталогам или файлам.
- Механизм [авторизации](#) пользователей для доступа к каталогу на основе HTTP-аутентификации ([mod_auth_basic](#)) и [digest-аутентификации](#) ([mod_auth_digest](#)).
- Ограничение доступа к определённым каталогам или всему серверу, основанное на [IP-адресах](#) пользователей.
- Запрет доступа к определённым типам файлов для всех или части пользователей, например запрет доступа к конфигурационным файлам и файлам баз данных.
- Существуют модули, реализующие авторизацию через [СУБД](#) или [PAM](#).

В некоторых МРМ-модулях присутствует возможность запуска каждого процесса Apache, используя различные [uid](#) и [gid](#) с соответствующими этим пользователям и группам пользователям.

Также существует механизм [suexec](#), используемый для запуска [скриптов](#) и [CGI](#)-приложений с правами и идентификационными данными пользователя.

Для реализации [шифрования](#) данных, передающихся между клиентом и сервером, используется механизм [SSL](#), реализованный через библиотеку [OpenSSL](#). Для удостоверения подлинности веб-сервера используются сертификаты [X.509](#).

Существуют внешние средства обеспечения безопасности, например [mod_security](#).

- [Интернационализация](#)

Начиная с версии 2.0 появилась возможность определения сервером [локали](#) пользователя. Сообщения об ошибках и событиях, посылаемые браузеру, теперь представлены на нескольких языках и используют [SSI](#)-технологию.

Также, можно реализовать средствами сервера отображение различных страниц для пользователей с различными локалами. Apache поддерживает множество кодировок, в том числе [Unicode](#), что позволяет использовать страницы, созданные в любых кодировках и на любых языках.

- Обработка событий

Администратор может установить собственные страницы и обработчики для всех [HTTP](#)-ошибок и событий, таких как 404 (Not Found) или 403 (Forbidden). В том числе существует возможность запуска [скриптов](#) и отображения сообщений на разных языках.

[SSI](#) - В версиях 1.3 и старше был реализован механизм Server Side Includes, позволяющий динамически формировать [HTML](#)-документы на стороне сервера. Управлением SSI занимается модуль [mod_include](#), включённый в базовую поставку Apache.

База данных – [Data Base](#) – совокупность данных, хранимых в соответствии со схемой данных, манипулирование с которыми осуществляют в соответствии с правилами средств модулирования данных.

База данных — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

Признаки:

- БД хранится и обрабатывается в вычислительной системе
- Данные БД логически структурированы для облегчения работы с ними (систематизация: выделение составных частей, построение связей между ними, типизация, семантика и допустимые операции)
- БД включает схему, или метаданные, описывающие ее логическую структуру в формальном виде (в соответствии с некоторой метамоделью)

Виды БД:

По модели данных:

- [иерархические](#)
- [объектные](#) или [объектно-ориентированные](#)
- [объектно-реляционные](#)
- [реляционные](#)
- [сетевые](#)
- функциональные

По системе хранения:

- традиционные – conventional DB – хранят данные во вторичной памяти
- [резидентные](#) – все данные на стадии исполнения находятся в оперативной памяти
- третичные – tertiary DB – хранят данные на отсоединяемых устройствах массового хранения

Для некоторых БД строятся [специализированные СУБД, либо доп. возможности СУБД](#):

- пространственные (spatial DB) – базы с пространственными свойствами сущности предметной области
- временные – поддерживают к-л аспект времени (не считая времени, определяемого пользователем)

По степени распределенности:

- централизованные (сосредоточенные) – полностью поддерживаемые на одном оборудовании

- распределенные (distributed DB) – на разных узлах комп. сети по к-л критерию. Среди них выделяют:
 - сегментированные – разделенные на части под управлением различных экземпляров СУБД по к-л критерию
 - тиражированные (реплицированные) – в них одни и те же данные разнесены под управление разных экземпляров СУБД
 - неоднородные (heterogeneous distributed DB) – фрагменты распределенной базы в разных узлах сети поддерживаются средствами более одной СУБД

По способам организации хранения:

- циклические – записывают новые данные заместо устаревших
- потоковые

SOLID

Хороший код:

- Масштабируемый, легко вносить изменения
- Порог вхождения в проект – новому человеку легко разобраться
- Простой, без усложнений
- **S – Single Responsibility Principle** – один объект – одна задача, одна зона ответственности

Декомпозиция на модули. Разделяем модель данных и поведение.

Иначе – путаница, трудно вносить изменения. Декомпозированный код легче читать, править, тестировать, мерджить.

God-object – делает много задач.

- **O – Open/Closed Principle** – программные сущности открыты для расширения и закрыты для изменения.

Новый функционал вводим за счет создания новых сущностей путем наследования, композиции, а не изменением старых.

Если и изменяем код, необходимо регрессионное тестирование.

- **L – Liskov's Substitution Principle** - функции, сущности, которые используют родительский тип, должны так же работать и с дочерними классами, при этом не должна нарушаться логика программы. Наследуемый класс должен дополнять, а не замещать поведение базового класса.
- **I – Interface Segregation Principle** – программные сущности не должны зависеть от методов, которые они не используют.

Код менее связанный, сущности не зависят от методов, которые к ним не относятся.

- **D – Dependency Inversion Principle** – модули высокого уровня не должны зависеть от модулей нижнего уровня, все они должны зависеть от абстракций, а абстракции не должны зависеть от деталей, наоборот – детали должны зависеть от абстракций.

Парадигма ООП

Процедурный подход – программе даются данные, она выполняет какие-то функции и возвращает какой-то результат.

В ООП (объектно-ориентированном программировании) есть класс, в нем есть объекты, у которых есть свойства. Объект – конкретный представитель класса со своими обозначенными свойствами. Объект может совершать некоторые действия – методы.

Объекты здесь не только классы (и объекты), но и сущности более высокого уровня – модули, пакеты, неймспейсы, сабсистемы, система, сабпакеты и т.д.

Основные концепции ООП (принципы):

1. Инкапсуляция

Сам класс – капсула, которая содержит свойства и методы для работы с этими свойствами.

В объект (класс) объединяются методы и данные.

Скрытие – private (можно использовать только внутри класса, использовать извне нельзя) и public (модификаторы доступа) (одна из трактовок инкапсуляции).

Итого, это и объединение методов и данных в один объект, и скрытие этих методов и данных от внешнего воздействия. Объект не должен изменяться ничем извне, кроме методов самого объекта (внутри).

Get, set – создаются, чтобы получать доступ к приватным свойствам (получать и изменять).

Пример: имя пользователя и пароль можно получать и менять, а ID – только получать.

Из инкапсуляции вытекает множество паттернов GRASP и GoF.

Lowcoplin

2. Наследование

Новые классы наследуют черты от родительского(-их) класса(-ов) и имеют собственные свойства.

Не можем унаследоваться от нескольких классов (в большинстве языков).

Конструктор т.ж. по умолчанию наследуется.

3. Полиморфизм

В общем, это способность функции работать с данными разных типов.

Есть два типа:

- Полиметрический (истинный)

Когда одна и та же функция с одним и тем же телом способна принимать в качестве параметра данные разных классов.

Пример: у нас есть несколько классов: родительский класс «человек», от него наследуется класс «рабочник», а уже от него – класс «программист». Всем классам нужно написать функцию приветствия, возвращающую «Привет» + «я» + название класса + имя объекта.

Принимать данные от каждого массива – плохая идея, поэтому мы применяем полиморфизм, чтобы каждый объект, унаследованный от класса «человек» смог поздороваться.

Создаем массив со всеми объектами, создаем функцию массового приветствия с аргументом массивом.

- ad-hoc (мнимый) – класс имеет несколько методов, например, работающих с разными типами данных, а так же класс, в котором мы делаем преобразование дочернего класса до родительского

Это приведение данных к тому типу, с которым работаем метод; перегрузка метода – когда он существует с одинаковыми названиями, но разными параметрами.

Есть высказывание, что все if можно заменить на полиморфизм. Он помогает в разы уменьшать размер программы.

Рефакторинг, coding by exception.

Еще сейчас выделяют 4 принцип – абстракция.

Помимо наследования существуют еще два способа взаимодействия классов:

- Композиция

Пример: есть класс автомобиль, внутри которого есть объект класса двигатель, а также массив объектов класса колесо. Двигатель и колеса не могут существовать отдельно от автомобиля.

- Агрегация

Все тот же автомобиль. Однако добавляется класс «елочка-освежитель». Он может существовать отдельно от класса «автомобиль». Тогда елочка будет отдельным свойством у класса «автомобиль», обращающимся к собственному классу – «освежитель». А в конструкторе объекта мы к нему обращаемся через свойство.

Абстрактные классы и интерфейсы

Интерфейс – как оглавление в учебнике. Он говорит, что нужно сделать, но не говорит как. Из него нельзя создать объект.

Абстрактные классы похожи на интерфейс, кроме того, в них можно определять абстрактные методы (аналоги методов в интерфейсе, т.е. без реализации), но в них можно создавать и обычные методы с реализацией и логикой. Класс, образованный от абстрактного, наследует все обычные методы, а т.ж. абстрактные.

generic – обобщение <ожидаемый тип данных>

Пример для закрепления: Dependency Injection (внедрение зависимостей) (паттерн)

У нас есть два слоя приложения:

1) работа с БД (получить, записать, изменить, удалить из репозитория). Реализации – через MongoRepository и MySqlRepository.

2) БЛ. Здесь вопрос: какой репозиторий выбрать? Лучше, чтобы сервисный слой не знал, с каким репозиторием работает. Т.е. мы делаем имплементацию БД извне, пр. на уровне конфигурации. Так нам не надо менять сервисный слой, достаточно поправить конфигурацию.

interface UserRepo

 getUsers

class UserMongoDBRepo implements UserRepo

class UserService

 userRepo: UserRepo

 constructor (userRepo: UserRepo):

 this.userRepo = userRepo

 filterUserByAge(age: number)

```

const users = this.UserRepo.getUsers
    // какая-то логика по фильтрации
    console.log(users)
const userService = new UserService(new UserMongoDBRepo)
userService.filterUserByAge (age=13)

```

Мы можем передать в конструктор аргумент new UserMySQLDBRepo, то есть извне, а сам сервис не трогали.

Еще пример с паттерном Singleton (нужно, чтоб было только одно подключение к БД):

```

class Database

url: string

private static instance: Database

constructor(url: string)

if Database.instance

    return Database.instance

this.url = Math.random()

Database.instance = this

const db1 = new Database()

const db2 = new Database()

console.log(db1.url)

console.log(db2.url)

```

Выведет два одинаковых рандомных числа, т.е. вход один.

Паттерны проектирования

П. пр. – это часто встречающееся решение (концепция) определенной проблемы при проектировании архитектуры программ.

Польза:

- проверенные решения (не надо изобретать велосипед)
- стандартизация кода (проще для понимания, меньше просчетов, а значит, и костылей)
- общий программистский словарь (удобство понимания)

Критика:

- костыль для языка с недостаточным уровнем абстракции («слабого» в данной ситуации)
- могут быть неэффективны, если применяются без адаптации под конкретный проект
- новички начинают «пишать» паттерны везде, даже если можно сделать проще

Классификация:

- Самые простые и низкоуровневые – идиомы – реализуемы в рамках одного языка.

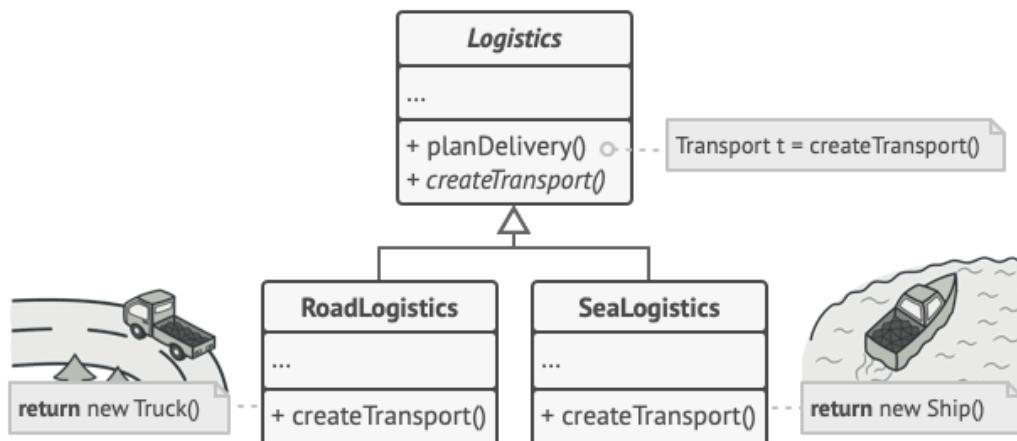
- Самые высокоуровневые – архитектурные паттерны – применимы для всех языков и служат для проектирования программ в целом, не отдельных элементов.
- Классификация по предназначению:
 - Порождающие – служат для быстрого создания объектов без внесения в программу лишних зависимостей
 - Структурные – показывают различные способы построения связей между объектами
 - Поведенческие – заботятся об эффективной коммуникации между объектами

Каталог паттернов:

- **Порождающие**
 - **Фабричный метод (Factory method)** – он же «виртуальный конструктор» - порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Решаемая проблема: если код задействует один класс (пр.: грузовик), а потом надо внедрить новый класс (пр.: пароход), то придется переписывать весь код, создавать условные операторы, которые работают в зависимости от класса.

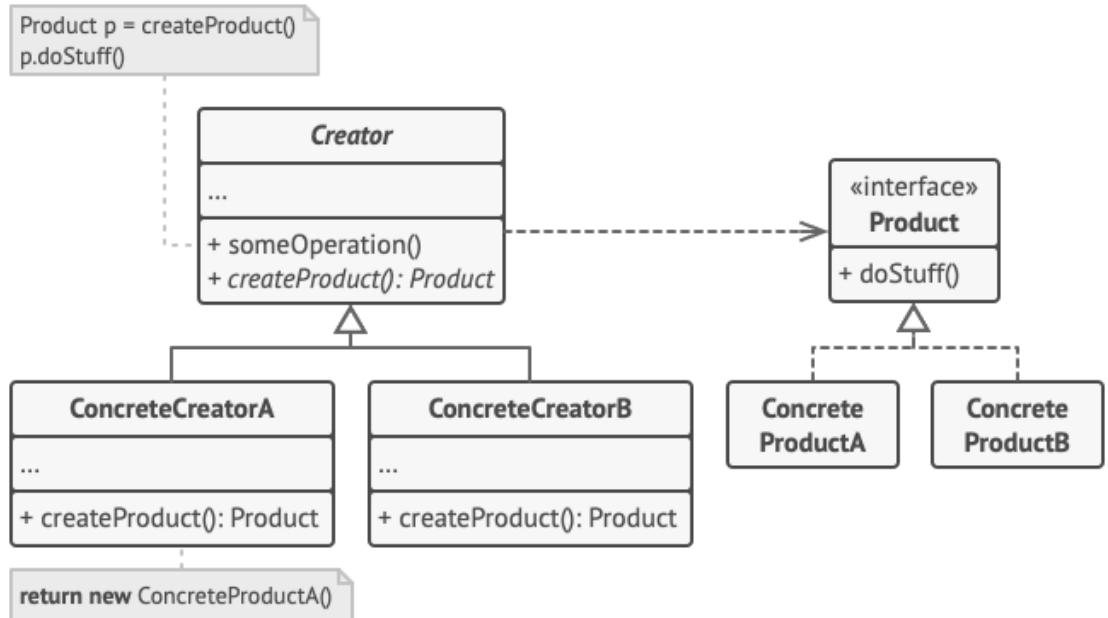
Решение: объекты создаются не напрямую, а через вызов фабричного метода (при этом фабричный метод использует все те же операторы, например, new).



Подклассы смогут производить объекты различных классов, следующих одному и тому же интерфейсу.

Например, классы Грузовик и Судно реализуют интерфейс Транспорт с методом доставить. Каждый из этих классов реализует метод по-своему: грузовики везут грузы по земле, а суда — по морю. Фабричный метод в классе «Дорожной Логистики» вернёт объект-грузовик, а класс «Морской Логистики» — объект-судно.

Для клиента фабричного метода нет разницы между этими объектами, так как он будет трактовать их как некий абстрактный Транспорт. Для него будет важно, чтобы объект имел метод доставить, а как конкретно он работает — не важно.

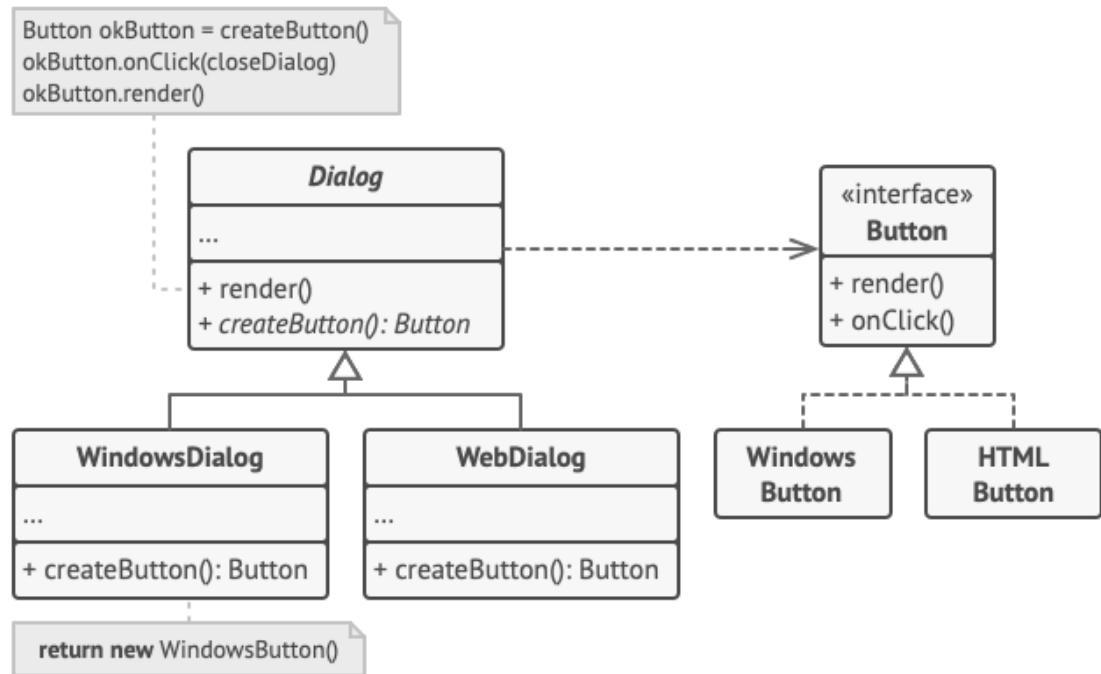


1. Продукт определяет общий интерфейс объектов, которые может произвести создатель и его подклассы.
2. Конкретные продукты содержат код различных продуктов, которые отличаются реализацией, но имеют общий интерфейс.
3. Создатель объявляет фабричный метод, который должен возвращать новые объекты продуктов. Важно, чтобы тип результата совпадал с общим интерфейсом продуктов. Обычно, ФМ – абстрактный, но может создавать и типовой продукт. Создатель, помимо создания продуктов, может выполнять и сторонние функции.
4. Конкретные создатели по-своему реализуют фабричный метод, производя те или иные конкретные продукты.

Фабричный метод можно и переписать так, чтобы возвращать существующие объекты из какого-то хранилища или кэша.

Пример:

В этом примере Фабричный метод помогает создавать кросс-платформенные элементы интерфейса, не привязывая основной код программы к конкретным классам элементов.



Фабричный метод объявлен в классе диалогов. Его подклассы относятся к различным операционным системам. Благодаря фабричному методу, вам не нужно переписывать логику диалогов под каждую систему. Подклассы могут наследовать почти весь код из базового диалога, изменяя типы кнопок и других элементов, из которых базовый код строит окна графического пользовательского интерфейса.

Базовый класс диалогов работает с кнопками через их общий программный интерфейс. Поэтому, какую вариацию кнопок ни вернул бы фабричный метод, диалог останется рабочим. Базовый класс не зависит от конкретных классов кнопок, оставляя подклассам решение о том, какой тип кнопок создавать.

Стоит применять:

- когда заранее не известны типы и зависимости объектов, с которыми должен работать код (код производства продуктов идет отдельно от остального кода, так что его можно расширять, не трогая основной код; например, чтобы добавить поддержку нового продукта, вам нужно создать новый подкласс и определить в нём фабричный метод, возвращая оттуда экземпляр нового продукта)
- когда вы хотите дать возможность пользователям расширять части вашего фреймворка или библиотеки (через наследование, создание и использование подклассов)
- когда вы хотите экономить системные ресурсы, повторно используя уже созданные объекты, вместо создания новых (т.к. в таком случае нужен метод, который будет и отдавать существующие объекты, и новые)

Шаги реализации:

1. Приведите все создаваемые продукты к общему интерфейсу.
2. В классе, который производит продукты, создайте пустой фабричный метод. В качестве возвращаемого типа укажите общий интерфейс продукта.

3. Затем пройдитесь по коду класса и найдите все участки, создающие продукты. Поочерёдно замените эти участки вызовами фабричного метода, перенося в него код создания различных продуктов.

4. В фабричный метод, возможно, придётся добавить несколько параметров, контролирующих, какой из продуктов нужно создать.

На этом этапе фабричный метод, скорее всего, будет выглядеть удручающе. В нём будет жить большой условный оператор, выбирающий класс создаваемого продукта. Но не волнуйтесь, мы вот-вот исправим это.

5. Для каждого типа продуктов заведите подкласс и переопределите в нём фабричный метод. Переместите туда код создания соответствующего продукта из суперкласса.

6. Если создаваемых продуктов слишком много для существующих подклассов создателя, вы можете подумать о введении параметров в фабричный метод, которые позволят возвращать различные продукты в пределах одного подкласса.

Например, у вас есть класс Почта с подклассами АвиаПочта и НаземнаяПочта, а также классы продуктов Самолёт, Грузовик и Поезд. Авиа соответствует Самолётам, но для НаземнойПочты есть сразу два продукта. Вы могли бы создать новый подкласс почты для поездов, но проблему можно решить и по-другому. Клиентский код может передавать в фабричный метод НаземнойПочты аргумент, контролирующий тип создаваемого продукта.

7. Если после всех перемещений фабричный метод стал пустым, можете сделать его абстрактным. Если в нём что-то осталось — не беда, это будет его реализацией по умолчанию.

Плюсы и минусы:

- + Избавляет класс от привязки к конкретным классам продуктов.
- + Выделяет код производства продуктов в одно место, упрощая поддержку кода.
- + Упрощает добавление новых продуктов в программу.
- + Реализует принцип открытости/закрытости.
- Может привести к созданию больших параллельных иерархий классов, так как для каждого класса продукта надо создать свой подкласс создателя.

Отношения с другими паттернами:

- Многие архитектуры начинаются с применения Фабричного метода (более простого и расширяемого через подклассы) и эволюционируют в сторону Абстрактной фабрики, Прототипа или Строителя (более гибких, но и более сложных).
- Классы Абстрактной фабрики чаще всего реализуются с помощью Фабричного метода, хотя они могут быть построены и на основе Прототипа.
- Фабричный метод можно использовать вместе с Итератором, чтобы подклассы коллекций могли создавать подходящие им итераторы.
- Прототип не опирается на наследование, но ему нужна сложная операция инициализации. Фабричный метод, наоборот, построен на наследовании, но не требует сложной инициализации.
- Фабричный метод можно рассматривать как частный случай Шаблонного метода. Кроме того, Фабричный метод нередко бывает частью большого класса с Шаблонными методами.

Примеры реализации – см. в [источнике](#).

```
from __future__ import annotations
from abc import ABC, abstractmethod

class Creator(ABC):
    """
        Класс Создатель объявляет фабричный метод, который должен
        возвращать объект
        класса Продукт. Подклассы Создателя обычно предоставляют
        реализацию этого
        метода.
    """

    @abstractmethod
    def factory_method(self):
        """
            Обратите внимание, что Создатель может также обеспечить
            реализацию
            фабричного метода по умолчанию.
        """

        pass

    def some_operation(self) -> str:
        """
            Также заметьте, что, несмотря на название, основная обязанность
            Создателя не заключается в создании продуктов. Обычно он
            содержит
            некоторую базовую бизнес-логику, которая основана на объектах
            Продуктов,
            возвращаемых фабричным методом. Подклассы могут косвенно
            изменять эту
            бизнес-логику, переопределяя фабричный метод и возвращая из
            него другой
            тип продукта.
        """

        # Вызываем фабричный метод, чтобы получить объект-продукт.
        product = self.factory_method()

        # Далее, работаем с этим продуктом.
        result = f"Creator: The same creator's code has just worked with {product.operation()}""

        return result

    """
        Конкретные Создатели переопределяют фабричный метод для того,
        чтобы изменить тип
        результирующего продукта.
    """

class ConcreteCreator1(Creator):
    """
        Обратите внимание, что сигнатура метода по-прежнему использует
        тип
        абстрактного продукта, хотя фактически из метода возвращается
        конкретный
        продукт. Таким образом, Создатель может оставаться независимым от
        конкретных
        классов продуктов.
    """

    def factory_method(self) -> Product:
        return ConcreteProduct1()

class ConcreteCreator2(Creator):
    def factory_method(self) -> Product:
        return ConcreteProduct2()
```

```

class Product(ABC):
    """
    Интерфейс Продукта объявляет операции, которые должны выполнять
    все
    конкретные продукты.
    """

    @abstractmethod
    def operation(self) -> str:
        pass


    """
    Конкретные Продукты предоставляют различные реализации интерфейса
    Продукта.
    """

class ConcreteProduct1(Product):
    def operation(self) -> str:
        return "[Result of the ConcreteProduct1]"

class ConcreteProduct2(Product):
    def operation(self) -> str:
        return "[Result of the ConcreteProduct2]"

def client_code(creator: Creator) -> None:
    """
    Клиентский код работает с экземпляром конкретного создателя,
    хотя и через
    его базовый интерфейс. Пока клиент продолжает работать с
    создателем через
    базовый интерфейс, вы можете передать ему любой подкласс
    создателя.
    """

    print(f"Client: I'm not aware of the creator's class, but it still works.\n"
          f"[{creator.some_operation()}", end="")

if __name__ == "__main__":
    print("App: Launched with the ConcreteCreator1.")
    client_code(ConcreteCreator1())
    print("\n")

    print("App: Launched with the ConcreteCreator2.")
    client_code(ConcreteCreator2())

```

Output.txt: Результат выполнения

```

App: Launched with the ConcreteCreator1.
Client: I'm not aware of the creator's class, but it still works.
Creator: The same creator's code has just worked with [Result of the ConcreteProduct1]

App: Launched with the ConcreteCreator2.
Client: I'm not aware of the creator's class, but it still works.
Creator: The same creator's code has just worked with [Result of the ConcreteProduct2]

```

- **Абстрактная фабрика (Abstract factory)**

Это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

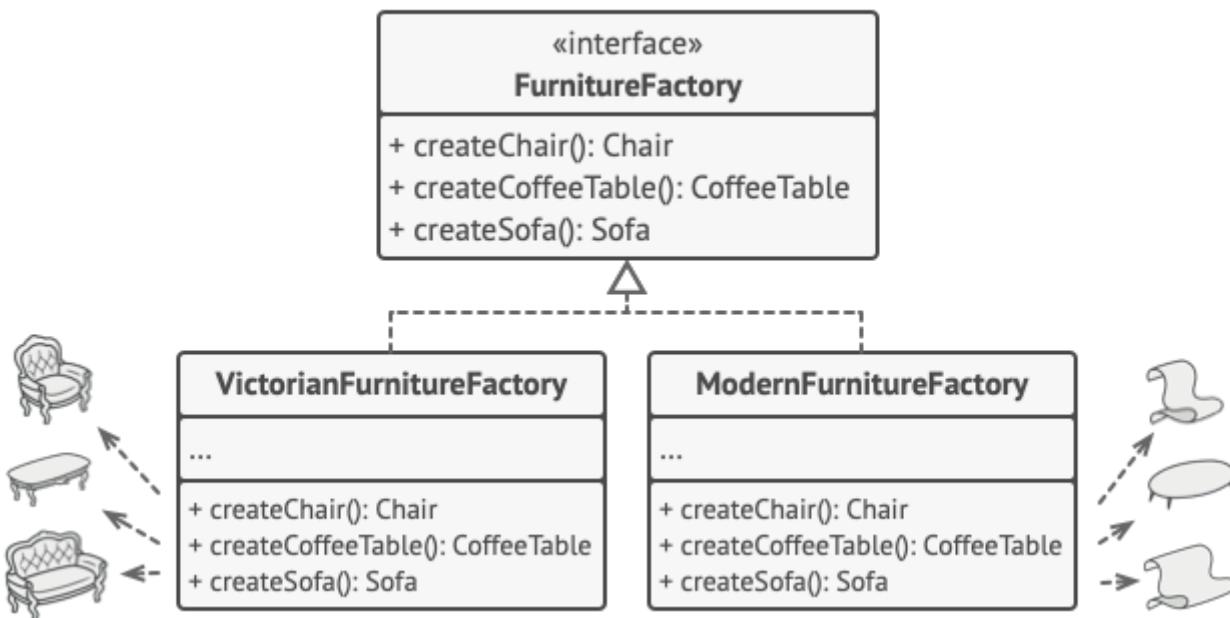
Решаемая проблема: пример – мебельный магазин. Есть:

- семейство зависимых продуктов (кресло+диван+столик)
- несколько вариаций этого семейства (стили модерн, классика и т.д.)

Нужен способ создавать объекты продуктов, чтобы они сочетались с другими продуктами того же семейства, не внося изменений в существующий код при создании новых продуктов или семейств.

Решение: выделяем общие интерфейсы для отдельных продуктов, составляющих семейства (у кресел свой одинаковый интерфейс и т.д.).

Создаем абстрактную фабрику – общий интерфейс, который содержит методы создания всех продуктов семейства. Эти операции должны возвращать абстрактные типы продуктов, представленные интерфейсами (кресла, диваны и т.д.)



Для каждой вариации продуктов мы должны создать свою собственную фабрику, реализовав абстрактный интерфейс. Фабрики создают продукты одной вариации (ФабрикаМодерн – КреслаМодерн, ДиванМодерн и т.д.).

Клиентский код должен работать как с фабриками, так и с продуктами только через их общие интерфейсы, что позволяет подавать в классы любой тип фабрики и производить любые продукты.

Кто же создает объекты конкретных фабрик, если клиентский код создает только интерфейсы самих фабрик? Обычно программа создает конкретный объект фабрики при запуске, причем тип фабрики выбирается, исходя из параметров окружения или конфигурации.

- Строитель (Builder)
- Прототип (Prototype)
- Одиночка (Singleton)
- Структурные
 - Адаптер (Adapter)
 - Мост (Bridge)
 - Компоновщик (Composite)
 - Декоратор (Decorator)
 - Фасад (Facade)
 - Легковес (Flyweight)
 - Заместитель (Proxy)
- Поведенческие
 - Цепочка обязанностей (Chain of Responsibility)
 - Команда (Command)
 - Итератор (Iterator)

- Посредник (Mediator)
- Снимок (Memento)
- Наблюдатель (Observer)
- Состояние (State)
- Стратегия (Strategy)
- Шаблонный метод (Template Method)
- Посетитель (Visitor)

JSON

JavaScript Object Notation – текстовый формат обмена данными, основанный на JavaScript. Используется в REST API. Альтернатива – XML.

Значения: числа, строки, массивы, JSON-объекты, литералы (лог. зн. true, false, null).

JSON-объект – неупорядоченное множество пар {"ключ" : значение}, взаимодействие с которыми происходит, как со словарями.

Ключ – название параметра, который мы передаем серверу. Он служит маркером для принимающей стороны запрос системы, чтобы она поняла, что мы ей отправили.

Ключ всегда строка. Строки в кавычках. Порядок – любой.

Обратиться к ключу можно через:

- get() – если ключа нет, то пустое значение (NoneType), также имеет второй передаваемый аргумент (возвращается, если ключа нет)
- dict[key] – если ключа нет, то KeyError

JSON-массив: [“MALE”, “FEMALE”].

Нет ключей, набор значений, обращение по номеру элемента. Нельзя менять местами данные – упорядоченное множество.

Well Formed JSON – синтаксически правильный.

Правила:

1. Данные написаны в виде пар «ключ: значение»
2. Данные разделены запятыми
3. Объект находится внутри {фигурных скобок}
4. Массив – внутри [квадратных]

Для проверки синтаксиса - JSON Validator, JSON Formatter (он еще и форматирует).

Бизнес-логика

Это правила работы программы, взаимодействия ее внутренних составляющих, а также взаимодействие со внешними элементами.

----- не по теме, но интересно -----

- [] How does the Internet work?
- [] How does a computer work?
- [] What is “back-end”?
- [] Hosting
- [] API
- [] Server
- [] DNS

- [] HTTP
- [] Browser
- [] Domain name
- [] About Python
- [] IDE (e.g. PyCharm, VS Code, Visual Studio, IntelliJ Idea)
- [] Terminal (PowerShell, Bash, Zsh)
- [] Framework (Django)
- [] Data Bases (PostgreSQL, MySQL, MS SQL, Azure Cloud, SQL; MongoDB, Redis, Cassandra, AWS, Firebase)
- [] Data Structures
- [] Authentication (OAuth 2.0, JWT; providers - Auth0, Firebase)
- [] Patterns of Programming
- [] Testing, QA (frameworks)
- [] Package Managers (NuGet, npm, yarn)
- [] Version Control Systems (git, GitHub, TFS, BitBucket)
- [] Web-services
- [] Network Protocols
- [] UI-frameworks
- [] OOP



Python

Here're some Python basics you need to know

About language:

▼ What is Python used for?

- web-development (server-side)
- software development
- mathematics
- system scripting

▼ What can Python do?

- Python can be used on a server to web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

▼ Why Python?

- Python works on different platforms, OS (Windows, Mac, Linux, Raspberry, Pi, etc.).
- Python has a simple syntax to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

▼ Gear: IDE, frameworks

- IDE (Integrated Development Environment): Thonny, PyCharm, Netbeans, Eclipse.
- Frameworks: Django.

▼ Python compared to other languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation (табуляция), using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

▼ Python and Functional Programming

Python is a multi-paradigm language. It supports imperative, object oriented, as well as functional programming approach.

Functional programming paradigm recommends dividing the programming logic into set of functions that only take input and produce output not that affects the output produced for a given input.

Features of Python which help in writing functional-style programs: lambda function, recursion, iterator, generator, list comprehension, the itertools module, map(), filter(), reduce().

Python and OOP: <https://www.knowledgehut.com/tutorials/python-tutorial/python-object-oriented-programming>

Variables

Variables are containers for strong data values; they are links to data that is stored in a computer's operative memory.

A variable is created (and its type is being declared) as soon as one assigns a value to it, and can be changed anytime.

▼ Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)
y = int(3)
z = float(3)
print(x)
print(y)
print(z)
```

→

3

3

3.0

▼ Get Type

▼ You can get the type of a variable by using type() function

```
x = str(3)
y = int(3)
z = float(3)
print(type(x))
print(type(y))
print(type(z))
```

→

<class 'str'>

<class 'int'>

<class 'float'>

▼ Names

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- Multi-names can be written in: camel case (myVariableName), Pascal case (MyVariableName), snake case (my_variable_name).

▼ Assign Multiple Values

▼ Many values to multiple variables

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

→

Orange

Banana

Cherry

▼ One value to multiple variables

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

→

Orange

Orange

Orange

▼ Unpack a collection

```
fruits = ["Apple", "Pear", "Kiwi"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

→

Apple

Pear

Kiwi

▼ Output Variables

Usually - with print().

▼ Also can place several variables in one line:

```
fruits = ["Apple ", "Pear ", "Kiwi"]
x, y, z = fruits
print(x+y+z)
```

→

Apple Pear Kiwi

▼ Global Variables

▼ These are variables that are created outside of a function. They can be used everywhere, by everyone, both outside functions and inside.

```
x = "Python"

def pyfunc_my():
    print(x+" is awesome!")

pyfunc_my()
```

→

Python is awesome!

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

- ▼ To create a global variable inside a function, you can use the `global` keyword.

```
def pyfunc_my():
    global x
    x = "Python"
    print(x+" is awesome!")

pyfunc_my()

print(x+" is not that hard.")
```

→

Python is awesome!

Python is not that hard.

- ▼ Also, use the `global` keyword if you want to change a global variable inside a function.

```
x = "Java"

print(x)

def pyfunc_my():
    global x
    x = "Python"
    print(x+" is awesome!")

pyfunc_my()

print(x+" is not that hard.")
```

→

Java

Python is awesome!

Python is not that hard.

Data Types:

Integers

These are simply whole numbers. Shortened as ‘int’. E.g.: 8 and -8. (also known as “numeric type”)

When a=25 the value is called “integer literal”. Function int(“25”) - converts str to int (but it’s not integer literal).

To divide a long number, like 1000000, one can use underscores(): 1_000_000.

Floating-Point Numbers

These are numbers with a decimal point. Shortened as ‘float’. E.g.: 8.0 and -0.9 and 1.3. Max = 2×10^{400} (2e400 → inf). (also known as “numeric type”)

When a = 2.5 - floating point literal, but float(“2.5”) (converts str to float) - no.

- ▼ 1000000.0 is 1_000_000.0 is 1e6 (E[xponential] notation, here equals 1×10^6).

```
>>> 20000000000000000000.0
2e+17 #+ means that the exponent 17 is positive
```

```
>>> 1e-4
0.0001 #raised in power -4
```

Complex Numbers

Have <real part>+<imaginary part>. E.g.: 2+3j will print (2+3j) (also known as “numeric type”).

▼ Operations with numbers

“+” - addition (e.g. $2 + 3 = 5$ and $1.0 + 3 = 4.0$)

“-” - subtraction (e.g. $3 - 2 = 1$ or $4.0 - 2 = 2.0$ or just -3)

“*” - multiplication (e.g. $2 * 2 = 4$ or $2.0 * 2 = 4.0$)

“/” - division (e.g. $\text{int}(4/2) = 2$ or $4/2 = 2.0$ or $9.0/3 = 3.0$ or $\text{int}(5.0/2) = 2$)

“//” - integer division or floor division (e.g. $5//2 = 2$ or $-3//2 = -2$, b.c. it round down the number)

“**” - power exponent (e.g. $2 ** 3 = 8$ or $9 ** 0.5 = 3.0$ or $2 ** -1 = 0.5$)

▼ “%” - modulus operator (e.g. 5%3 = 2 or 6%3 = 0)

Things get a little tricky when you use the `%` operator with negative numbers:

```
>>>
```

```
>>> 5 % -3  
-1
```

```
>>> -5 % 3  
1
```

```
>>> -5 % -3  
-2
```

Although potentially shocking at first glance, these results are the product of a well-defined behavior in Python. To calculate the remainder `r` of dividing a number `x` by a number `y`, Python uses the equation `r = x - (y * (x // y))`.

For example, to find `5 % -3`, Python first finds `(5 // -3)`. Since `5 / -3` is about `-1.67`, that means `5 // -3` is `-2`. Now Python multiplies that by `-3` to get `6`. Finally, Python subtracts `6` from `5` to get `-1`.

▼ Python Assignment Operators

- “`+=`” — example: `x += 3` — same as: `x = x + 3`
- “`-=`” — example: `x -= 3` — same as: `x = x - 3`
- “`*=`” — example: `x *= 3` — same as: `x = x * 3`
- “`/=`” — example: `x /= 3` — same as: `x = x / 3`
- “`%=`” — example: `x %= 3` — same as: `x = x % 3`
- “`//=`” — example: `x //= 3` — same as: `x = x // 3`
- “`**=`” — example: `x **= 3` — same as: `x = x ** 3`
- “`&=`” — example: `x &= 3` — same as: `x = x & 3`
- “`|=`” — example: `x |= 3` — same as: `x = x | 3`
- “`^=`” — example: `x ^= 3` — same as: `x = x ^ 3`
- “`<<=`” — example: `x <<= 3` — same as: `x = x << 3`

- “`>>=`” — example: `x >>= 3` — same as: `x = x >> 3`

▼ Other Operator Groups

- Comparison: `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Logical: and, or, not.
- Identity: `is`, `is not`.
- Membership: `in`, `not in`.

▼ Bitwise

Bitwise

<u>Aa</u> Operator	≡ Name	≡ Description	📅 Date
<u>May spa</u>	may spa	Summer prep: lilac, lilly of the vally, cherry blossom vibes	@May 21, 2023
<u>March spa</u>	march spa	Women's day prep: only flowers, bubbles and bright colors	@March 5, 2023
<u>December</u>	December spa	New Year prep: vanilla, gingerbread cookies, choco and shiny winter shimmer	@December 25, 2022
<u>June spa</u>	june spa	Birthday prep: sweet cake, bright sparkles and bubbles	@June 4, 2023
<u>October</u>	October spa	Halloween prep: pumpkin'n'autumn vibes, cozy and funny	@October 30, 2022
<u>February</u>	February spa	Valentine's day prep: wine, movies, strawberries in chocolate mood	@February 12, 2023
<u>September</u>	september spa	Autumn vibes: apple pie, cinnamon roll, pancakes with maple syrup	@September 23, 2023

▼ Integer and Floating Point Methods

- ▼ `num.is_integer()` - returns True if int

```
num = 5.6
print(num.is_integer())
num1 = 5.0
print(num1.is_integer())
```

→

False

True

▼ f-string for numbers

```
>>> n = 7.125  
>>> f"The value of n is {n}"  
'The value of n is 7.125'
```

Those curly braces support a simple [formatting language](#) that you can use to alter the appearance of the value in the final formatted string.

```
>>> n = 7.125  
>>> f"The value of n is {n:.2f}"  
'The value of n is 7.12'
```

```
>>> n = 1234567890  
>>> f"The value of n is {n:,}"  
'The value of n is 1,234,567,890'
```

The specifier `, .2f` is useful for displaying currency values:>>>

```
>>> balance = 2000.0  
>>> spent = 256.35  
>>> remaining = balance - spent  
  
>>> f"After spending ${spent:.2f}, I was left with ${remaining:.2f}"  
'After spending $256.35, I was left with $1,743.65'
```

Another useful option is `%` , which is used to display percentages.

```
>>> ratio = 0.9  
>>> f"Over {ratio:.1%} of Pythonistas say 'Real Python rocks!'"  
"Over 90.0% of Pythonistas say 'Real Python rocks!'"  
  
>>> # Display percentage with 2 decimal places
```

```
>>> f"Over {ratio:.2%} of Pythonistas say 'Real Python rocks!'"  
"Over 90.00% of Pythonistas say 'Real Python rocks!'"
```

Strings

These are sequences of character data. Shortened as ‘str’. E.g.: “Hey there!” or “123” or ‘’. (also known as “text type”)

If needt to include ‘ or “ in a str - “st’r”. Other - ‘st”r’.

▼ Escape Sequences ()

\' - leaves ‘ in ‘str’

\\" - leaves “ in “str”

\<new line> - newline is ignored, terminates input line

\\\ - leaves \ (backslash)

\a - ASCII Bell (**BEL**) character - nothing will be printed

\b - ASCII Backspace (**BS**) character - all in one line

\f - ASCII Formfeed (**FF**) character - new line saving all spaces

\n - new line saving all spaces

\t - tab space (horizontal)

\r - return to the begginigs of a str

\xxxx - Unicode character with 16-bit hex value **xxxx**

\Uxxxxxxxxx - Unicode character with 32-bit hex value **xxxxxxxx**

\v - ASCII Vertical Tab (**VT**) character - tab space (vertical)

\ooo - Character with octal value **ooo**

\xhh - Character with hex value **hh**

\N{name} - Character named ‘name’ in the Unicode database

Raw String - prefix ‘r’ ot ‘R’ before str - leaves all the escapes. E.g.: r’Hel\nlo’ → Hel\nlo.

Tripple-quoted str - tripple quotes enable ‘ and “ and newlines be leaft, all the escapes stay.

E.g.: “”” I’m so in love with \n\t YOU!””” →

I’m so in love with

YOU!

▼ Slicing strings

```
x = 'A fat cat sat on a mat'  
print(x[2:21:2])  
print(x[::-1])
```

→

ftctsto a
tam a no tas tac taf A

▼ String methods

▼ str.capitalize() - converts the first character to upper case

```
b = 'hey, honey!'  
print(b)  
print(b.capitalize())
```

→

hey, honey!
Hey, honey!

▼ str.casefold() - converts string into lower case

```
b = 'hey, honey!'  
print(b.upper())  
print(b.casefold())
```

→

HEY, HONEY!
hey, honey!

▼ str.center() - returns a centered string

```
b = 'hey, honey!'  
print(b.center(110, '.'))
```

→

.....hey, honey!.....

- ▼ str.count() - returns the number of times a specified value occurs in a string

```
b = 'hey, honey!'
print(b.count('h'))
print(b.count('y'))
```

→

2
2

- ▼ str.encode() - returns an encoded version of a string

```
getdefaultencoding()
# utf-8

my_string = 'кот cat'

type(my_string)
# str

my_string.encode()
# b'\xd0\xba\xd0\xbe\xd1\x82 cat'

my_string.encode('ascii')
# UnicodeDecodeError

my_string.encode('ascii', errors='ignore')
# b' cat'

my_string.encode('ascii', errors='replace')
# b'??? cat'

my_string.encode('ascii', errors='xmlcharrefreplace')
# b'кот cat'

my_string.encode('ascii', errors='backslashreplace')
```

```

# b'\u043a\u043e\u0442 cat'

my_string.encode('ascii', errors='namereplace')

# \N{CYRILLIC SMALL LETTER KA}\N{CYRILLIC SMALL LETTER O}\N{CYRILLIC SMALL LETTER TE} cat'

surrogated = '\udcd0\udcba\udcd0\udcbe\udcd1\udc82 cat'

surrogated.encode()

# UnicodeEncodeError

surrogated.encode(errors='surrogateescape')

# \xd0\xba\xd0\xbe\xd1\x82 cat'

surrogated.encode(errors='surrogatepass')

# \xed\xb3\x90\xed\xb2\xba\xed\xb3\x90\xed\xb2\xbe\xed\xb3\x91\xed\xb2\x82 cat'

```

- ▼ str.endswith() - returns True if the string ends with the specified value

```

b = 'hey, honey!'
print(b.endswith('!'))
print(b.endswith('honey!'))
print(b.endswith('world!'))

```

→

True

True

False

- ▼ str.expandtabs() - sets the tab size of the string

```

header_table = 'Name!\tRace\tWeapon\tMount'
person1 = 'Tiana\tElf\tSword\tHell Horse'
print(header_table.expandtabs(15))
print(person1.expandtabs(15))

```

→

Name!	Race	Weapon	Mount
Tiana	Elf	Sword	Hell Horse

▼ str.find() - searches the string for a specified value and returns its position

```
lyrics = "I'm completely gone from your mind\nSoon as you realize\nYou let me go  
lighter than air"  
word = lyrics.find('air')  
print(lyrics)  
print('''\nThe word "air" is number''' +str(word)+"in the poem.")
```

→

I'm completely gone from your mind
Soon as you realize
You let me go lighter than air

The word "air" is number82in the poem.

▼ str.format() - formats specified values in a string

```
def cashier(x):  
    return """Here's your check for ${}.""".format(x)  
  
print(cashier(5))
```

→

Here's your check for \$5.

▼ str.format_map() - formats specified values in a string

```
point = {'x':4, 'y':-5}  
print('{x} {y}'.format_map(point))  
  
point = {'x':4, 'y':-5, 'z': 0}  
print('{x} {y} {z}'.format_map(point))
```

→

```
4 -5  
4 -5 0
```

The `format_map()` method is similar to `str.format(mapping)` except that `str.format(**mapping)` creates a new dictionary whereas `str.format_map(mapping)` doesn't.**

- ▼ `str.index()` - searches the string for a specified value and returns its position

```
quote = "Everything is hard before it's easy."
print(quote.find('easy'))
```

→

31

- ▼ `str.isalnum()` - returns True if all characters in a string are alphanumeric

```
year = "1917"
event = "1917 revolution"
print(year.isalnum())
print(event.isalnum())
```

→

True

False

- ▼ `str.isalpha()` - returns True if all characters in a string are in the alphabet

```
army = "army"
red_army = "red army"
event = "1917 revolution"
print(army.isalpha())
print(red_army.isalpha())
print(event.isalpha())
```

→

True

False

False

- ▼ `str.isascii()` - returns True if all characters in a string are ASCII characters

```
".isascii() # True  
'wz'.isascii() # True  
'w z'.isascii() # True  
'фы'.isascii() # False  
'wzфы'.isascii() # False
```

- ▼ str.isdecimal() - returns True if all characters in a string are decimal

```
year = "1917"  
event = "1917 revolution"  
print(year.isdecimal())  
print(event.isdecimal())
```

→

```
True  
False
```

- ▼ str.isdigit() - returns True if all characters in a string are digits

```
".isdigit() # False  
' '.isdigit() # False  
'!@#'.isdigit() # False  
'abc'.isdigit() # False  
'123'.isdigit() # True  
'abc123'.isdigit() # False
```

- ▼ str.isidentifier() - returns True if the string is an identifier

```
'continue'.isidentifier() # True  
'cat'.isidentifier() # True  
'function_name'.isidentifier() # True  
'ClassName'.isidentifier() # True  
'_'.isidentifier() # True  
'number1'.isidentifier() # True
```

```
'1st'.isidentifier() # False
```

```
'*'.isidentifier() # False
```

- ▼ str.islower() - returns True if all characters in teh string are lower case

```
year = "1917"
event = "1917 revolution"
rev = "revolution"
print(year.islower())
print(event.islower())
print(rev.islower())
```

→

False

True

True

- ▼ str.isnumeric() - returns True if all characters in the string are numeric

```
".isnumeric() # False
```

```
'a'.isnumeric() # False
```

```
'0'.isnumeric() # True
```

```
'10'.isnumeric() # True
```

```
'½'.isnumeric() # True
```

```
'XII'.isnumeric() # True
```

- ▼ str.isprintable() - returns True if all characters in the string are printable

```
".isprintable() # True
```

```
' '.isprintable() # True
```

```
'1'.isprintable() # True
```

```
'a'.isprintable() # True
```

```
'█'.isprintable() # False (Group Separator)
```

```
'█'.isprintable() # False (Escape)
```

- ▼ str.isspace() - retirns True if all characters in the string are whitespaces

```
year = "19 17"
space = "      "
print(year.isspace())
print(space.isspace())
```

→

False

True

- ▼ str.istitle() - returns True if the string follows the rules of a title

```
title1 = "The Lonely Cat"
title2 = "The lonely cat"
title3 = "THE LONELY CAT"
title4 = "the lonely cat"
print(title1.istitle())
print(title2.istitle())
print(title3.istitle())
print(title4.istitle())
```

→

True

False

False

False

- ▼ str.isupper() - returns True if all the characters in the string are upper case

```
title1 = "The Lonely Cat"
title2 = "The lonely cat"
title3 = "THE LONELY CAT"
title4 = "the lonely cat"
print(title1.isupper())
print(title2.isupper())
print(title3.isupper())
print(title4.isupper())
```

→

```
False  
False  
True  
False
```

▼ `iterable.join()` - converts elements of an iterable into a string

```
list = ["like", "love", "hate"]  
x = '-'.join(list)  
print(x)  
tuple = ("like", "love", "hate")  
y = '-'.join(tuple)  
print(y)  
dict = {"me":"love", "you":"like", "she":"hate"}  
z = '-'.join(dict)  
print(z)
```

→

```
like-love-hate  
like-love-hate  
me-you-she
```

▼ `str.ljust()` - returns a left justified version of the string

```
quote = "All we need is love"  
print(quote.ljust(30, 'e'))
```

→

```
All we need is loveeeeeeeeeeee
```

▼ `str.lower()` - converts the string into lower case

```
quote = "All we need is love"  
print(quote.lower())
```

→

```
all we need is love
```

▼ `str.lstrip()` - returns a left trim version of the string

```
quote = "All we need is love"
print(quote.lstrip('All'))
print((quote.lstrip('we'))))
print((quote.lstrip('need'))))
print((quote.lstrip('is'))))
```

→

```
we need is love
All we need is love
All we need is love
All we need is love
```

- ▼ `srt.maketrans()` - returns a translation table to be used in translations

```
trans_table = str.maketrans({
    'a': 'b',
    'r': 't',
    'z':None,
})
# {97: 'b', 114: 't', 122: None}

trans_table = str.maketrans('ar', 'bt', 'z')
# {97: 98, 114: 116, 122: None}

'arroz'.translate(trans_table)
# 'btto'
```

- ▼ `str.partition()` - returns a tuple where the string is parted into three parts

```
quote = "All we need is love"
print(quote.partition(' '))
print((quote.partition('need'))))
empty = ''
print(empty.partition('.')))
numb = '.2'
print((numb.partition('.'))))
```

→

```
('All', '', 'we need is love')
('All we ', 'need', ' is love')
(' ', ' ', ' ')
(' ', '!', '2')
```

- ▼ str.replace() - returns a string where a specified value is replaced with a specified value

```
quote = "All we need is love"
print(quote.replace('love', 'money'))
```

→

All we need is money

- ▼ str.rfind() - searches the string for a specified value and returns the last position of where it was found

```
quote = "All we need is love"
print(quote.rfind('e'))
```

→

18

- ▼ str.rindex() - searches the string for a specified value and returns the last position of where it was found

```
quote = "All we need is love"
print(quote.rindex('l'))
```

→

15

- ▼ str.rjust() - returns a right justified version of the string

```
quote = "All we need is love"
a = quote.rjust(75)
print(a, '- "The Beatles" sang.')
```

→

All we need is love - "The Beatles" sang.

- ▼ str.rpartition() - returns a tuple where the string is parted into three parts

```
quote = "All we need is love"
print(quote.rpartition('love'))
```

→

('All we need is ', 'love', '')

- ▼ str.rsplit() - splits the string at the specified separator, and returns a list

```
quote = "All we need is love"
print(quote.rsplit('is'))
```

→

['All we need ', ' love']

- ▼ str.rstrip() - returns a right trim version of the string

```
quote = "All we need is love"
print(quote.rstrip('e'))
```

→

All we need is lov

- ▼ str.split() - splits the string at the specified separator, and returns a list

```
quote = "All you and I need is love and health"
print(quote.split('and'))
```

→

['All you ', ' I need is love ', ' health']

▼ str.splitlines() - splits the string at line breaks and returns a list

```
quote = "All you and I need \nis love and health"  
print(quote.splitlines())  
print(quote.splitlines(keepends=True))
```

→

```
['All you and I need ', 'is love and health']  
['All you and I need \n', 'is love and health']
```

▼ str.startswith() - returns True if a string starts with a specified value

```
quote = "All you and I need \nis love and health"  
print(quote.startswith('All'))
```

→

```
True
```

▼ str.strip() - returns a trimmed version of a string

```
print('love'.strip('l'))  
print('love'.strip('e'))  
print('love'.strip('o'))
```

→

```
ove  
lov  
love
```

▼ str.swapcase() - swaps cases

```
print('No love allowed'.swapcase())  
print('cAPS LOCK LEAP'.swapcase())
```

→

nO LOVE ALLOWED

Caps lock leap

- ▼ str.title() - converts the first character of each word into upper case

```
print('No love allowed'.title())
print('cAPS LOCK LEAP'.title())
```

→

No Love Allowed

Caps Lock Leap

- ▼ str.translate() - returns a translated string

```
trans_table = str.maketrans({'л':'к', 'д':'а', '-':None})
replaced = 'лошка -ест сметану'.translate(trans_table)
print(replaced)
```

→

кошка ест сметану

- ▼ str.upper() - converts a string into upper case

```
song = 'Love the way \nyou lie'
print(song.upper())
```

→

LOVE THE WAY

YOU LIE

- ▼ str.zfill() - fills the string with a specified number of 0 values at the beginning

```
number = '13'
specific_number = number.zfill(10)
print(specific_number)
```

→

0000000013

Boolean Type

Also called as Boolean Context and “Truthiness”, Boolean Type means that the value of an object of this type may be True or False.

They are keywords, or expressions, more precisely.

▼ E.g.:

```
def love_test(petals):
    if petals % 2 == 0:
        return True
    else:
        return False

print(love_test(5))
print(type(love_test(5)))
```

→

```
False
<class 'bool'>
```

True = 1, False = 0

▼ Counting “the” in a poem

```
>>> lines = """\
... He took his vorpal sword in hand;
...       Long time the manxome foe he sought-
... So rested he by the Tumtum tree
...       And stood awhile in thought.
...
>>> line_list = lines.splitlines()
>>> "the" in line_list[0]
False
>>> "the" in line_list[1]
True
>>> 0 + False + True # Equivalent to 0 + 0 + 1
1
>>> ["the" in line for line in line_list]
[False, True, True, False]
>>> False + True + True + False
2
>>> len(line_list)
```

```
4
>>> 2/4
0.5
```

Shorter:

```
>>> lines="""\
... He took his vorpal sword in hand;
...     Long time the manxome foe he sought-
... So rested he by the Tumtum tree
...         And stood awhile in thought.
... """".splitlines()
>>> sum("the" in line.lower() for line in lines) / len(lines)
0.5
```

truth tables

short-circuit evaluation

The operator “**not**” (“not True” or “not False”). Another operator - “**and**” - takes two arguments (A and B), and means that the expression is False until both A and B are True. One more operator - “**or**” - is True until either A or B is True.

Comparison operators: “==” (equality); “!=” (inequality), “<” (less than, strict); “>” (greater than, strict); “<=” (less or equal, not strict), “>=” (greater or equal, not strict).

Can’t compare: str and int, dict, set. List, tuple can be compared as they are ordered lexicographically.

The operators “is” and “is not” check if A is the same object as B, or not.

The operator “in” checks an object from membership in somewhere (e.g. array).

One can chain operators: 1<3<7. Or mixed: a = 0 \ a is a<1 → True.

“None” is always False. All numbers are True, except for 0.

▼ One more value, or object in this case, evaluates to `False`, and that is if you have an object that is made from a class with a `__len__` function that returns `0` or `False`:

```
class myclass():
    def __len__(self):
```

```
return 0  
  
myobj = myclass()  
print(bool(myobj))
```

Functions

▼ Parametr or argument?

From a function's perspective:

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called.

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

▼ Arbitrary arguments

▼ If ikd how many args - *args:

```
def guests(*guests):  
    print(guests[0]+" was the first to come to my BD party!")  
  
guests("Jamie", "Anne", "Lucy", "Christoph")
```

→

Jamie was the first to come to my BD party!

▼ Keyword args

```
def family(mum, dad, me):  
    print("Hi! My name is "+me+". My mum's name is "+mum+". My dad's name is "+dad  
+".")  
family(mum = "Emily", dad="Peter", me="Clair")
```

→

Hi! My name is Clair. My mum's name is Emily. My dad's name is Peter.

▼ Arbitrary keyword args

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: `**` before the parameter name in the function definition.

▼ This way the function will receive a *dictionary* of arguments, and can access the items accordingly:

```
def detective_dairy(**info):
    print("I've got a client whose last name is"+info["lname"]+".\n")
    print("She told me to spy after her husband, "+info["hname"]+", as she thought he was cheating on her.\n")
    print("Yesterday morning I saw him leave for work, hiding "+info["present1"]+
" under suit coat.\n")
    print("In the afternoon he left his office and went to "+info["place1"]+
".\n")
    print("He came back with "+info["present2"]+".\n")
    print("Then he got a taxi to "+info["place2"]+".\n")
    print("He met "+info["person1"]+" at the entrance and greeted her as a close friend.\n")
    print("In the restaurant "+info["person2"]+" joined them.\n")
    print("After that they went to "+info["place3"]+".\n")
    print("Soon after the wife came to the same place. She realized that her husband prepared "+info["event"]+" for "+info["person3"]+".\n")
    print("That was an interesting story, even though I had to work more as "+info["proff"]+".")

print("\nHere's the first story:\n.....\n")
detective_dairy(lname="Clarson", hname="Ray", present1="a red box", place1="a flower shop", present2="a huge bouquet of red roses", place2="restaurant", person1="a red-haired girl", person2="a brightly dressed brunette", place3="his house", event="a birthday party", person3="her", proff="a photographer")

print("\nHere's the second story:\n.....\n")
detective_dairy(lname="Lanson", hname="Sam", present1="a black box", place1="a store", present2="a medium package", place2="a park", person1="a man in a blue suit", person2="a man in a white suit", place3="his house", event="a trap", person3="her", proff="a police officer")

print("\n.....")
print("\nIndeed, how a few details can change the whole picture! Fascinating, isn't it?")
```

→

Here's the first story:.....

I've got a client whose last name isClarson.

She told me to spy after her husband, Ray, as she thaught he was cheating on her.

Yesterday morning I saw him leave for work, hiding a red box under suit coat.

In the afternoon he left his office and went to a flower shop.

He came back with a huge bouquet of red roses.

Then he got a taxi to restaurant.

He met a red-haired girlat the entrance and greeted her as a close friend.

In the restaurant a brightly dressed brunettejoined them.

After that they went to his house.

Soon after the wife came to the same place. She realized that her husband prepared a birthday partyforher.

That was an interesting story, even though I had to work more as a photographer.

Here's the second story:.....

I've got a client whose last name isLanson.

She told me to spy after her husband, Sam, as she thaught he was cheating on her.

Yesterday morning I saw him leave for work, hiding a black box under suit coat.

In the afternoon he left his office and went to a store.

He came back with a medium package.

Then he got a taxi to a park.

He met a man in a blue suitat the entrance and greeted her as a close friend.

In the restaurant a man in a white suitjoined them.

After that they went to his house.

Soon after the wife came to the same place. She realized that her husband prepared a trapforher.

That was an interesting story, even though I had to work more as a police officer.

.....

Indeed, how a few details can change the whole picture! Fascinating, isn't it?

▼ Default parametr value

```
def attendant_country(country=" Russia"):
    print("- Hi! Where are you from?")
    print("- Hi! I'm from"+country+". Great conference, isn't it?")
    if country == "Russia":
        print("- Yes! And which city are you from?")
    else:
        print("- Indeed! How do you like our city? Have you been sightseeing?")

print("-----Dialogue 1-----")
attendant_country()
print()
print("-----Dialogue 2-----")
attendant_country(country=" the USA")
```

→

-----Dialogue 1-----

- Hi! Where are you from?
- Hi! I'm from Russia. Great conference, isn't it?
- Yes! And which city are you from?

-----Dialogue 2-----

- Hi! Where are you from?
- Hi! I'm from the USA. Great conference, isn't it?
- Indeed! How do you like our city? Have you been sightseeing?

▼ Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

▼ E.g. if you send a List as an argument, it will still be a List when it reaches the function

```
def words(wordlist):
    for x in wordlist:
        print(x)

vocab = ["a cat", "a mouse", "a house"]
words(vocab)
```

→

a cat
a mouse
a house

▼ return

▼ To let a function return a value, use the `return` statement:

```
def myfunc(x):
    return x**5

print(myfunc(1))
print(myfunc(2.1))
print(myfunc(-3))
```

→

1
40.84101000000001
-243

▼ pass

`function` definitions cannot be empty, but if you for some reason have a `function` definition with no content, put in the `pass` statement to avoid getting an error.

▼ Recursion

Recursion means that a function can call itself, which enables us to loop through data to reach a result.

▼ Example

In this example, `tri_recursion()` is a function that we have defined to call itself ("recurse"). We use the `k` variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

```
def tri_recursion(k):
    if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
    else:
        result = 0
```

```
    return result

print("Recursion Example Results")
tri_recursion(6)
```

→

Recursion Example Results

```
1
3
6
10
15
21
```



▼ Explanation

Try tracing the function with a pencil and paper. In this case, the print statement inside the function may be a bit misleading.

Consider this part of the program,

```
if(k>0):
    result = k+tri_recursion(k-1)
    ...
```

From here,

```
tri_recursion(6) = 6 + tri_recursion(5)
```

So to get the result for `tri_recursion(6)` we must get the result of `tri_recursion(5)`. Following this logic, the problem reduces to:

```
tri_recursion(6)
= 6 + tri_recursion(5)
= 6 + 5 + tri_recursion(4)
= 6 + 5 + 4 + tri_recursion(3)
= 6 + 5 + 4 + 3 + tri_recursion(2)
= 6 + 5 + 4 + 3 + 2 + tri_recursion(1)
= 6 + 5 + 4 + 3 + 2 + 1 + tri_recursion(0)
```

Now notice that 0 is not greater than 0 so the program moves to the body of the else clause:

```
else:
    result = 0
...
```

Which means `tri_recursion(0) = 0`. Therefore:

```
tri_recursion(6)
= 6 + 5 + 4 + 3 + 2 + 1 + tri_recursion(0)
= 6 + 5 + 4 + 3 + 2 + 1 + 0
= 21
```

Points to note:

1. In running this program, `k` is never equal to `1`, infact it is impossible.
2. It is misleading to think of control flow in terms of "the compiler moving across a program". The compiler doesn't do anything during execution (JIT is a different matter). It is better to think in terms of control flow / order of execution in procedural languages, equationally in functional programming and relations in logic programming.

▼ Lambda

It's an anonymous function, that can take any number of arguments, but can only have one expression.

▼ Syntax: `lambda arguments : expression`

```
x = lambda a, b : a/b
print(x(2, 3))
print(x(5, -10))
```

→

0.6666666666666666

-0.5

▼ Lambda can be used inside another function

```
def myfunc(n):
    return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

→

22

33

In Python there are some **Built-in functions:**

▼ Math

▼ `abs()` - returns absolute value of a number

```
print(abs(-26))
print(abs(0.2))
print(abs(83))
print(abs(-3.09))
```

→

26

0.2

83

3.09

- ▼ `divmod()` - returns quotient and remainder of integer division

```
print(divmod(45, 5))
print(divmod(33, 2))
```

→

(9, 0)

(16, 1)

- ▼ `max()` - returns the largest of given arguments or items in an iterable

```
list = [-3, 0, 1, 2, 4, 5, 7, 8]
print(max(list))
```

→

8

- ▼ `min()` - returns the smallest of given arguments or items in an iterable

```
list = ['lol', 'love']
list1 = ['a', 'b']
list2 = ['1', '2', 'a', 'b']
print(min(list))
print(min(list1))
print(min(list2))
```

→

lol

a

1

```
list3 = [1, 2, 'a', 'b']
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print(min(list3))

TypeError: '<' not supported between instances of 'str' and 'int'

```
tuple = (1, 2, 'love', 'a')
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3, in <module>
print(min(tuple))

TypeError: '<' not supported between instances of 'str' and 'int'

```
dict = {1:"May", "cat":13, 2:19, "mouse":"grey"}
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print(min(dict))

TypeError: '<' not supported between instances of 'str' and 'int'

```
tuple = ('love', 'a')
dict = {"mouse":"grey"}
print(min(tuple))
print(min(dict))
```

→

a

mouse

▼ pow() - raises a number to a power

```
print(pow(4, 2))
print(pow(-0.3, 3))
```

→

```
16  
-0.02699999999999996
```

▼ round() - rounds floating-point value

```
print(round(5.3))  
print(round(5.5))  
print(round(5.6))  
print(round(-5.3))  
print(round(-5.5))  
print(round(-5.6))
```

→

```
5  
6  
6  
-5  
-6  
-6
```

▼ sum() - sums the items of an iterable

```
list = [1, 2, -3, 5]  
tuple = (9, 8, 0.5)  
print(sum(list))  
print(sum(tuple))
```

→

```
5  
17.5
```

```
dict = {1:2, 7:4}  
print(sum(dict))
```

→ sums only keys

8

▼ Type Conversion

- ▼ `ascii()` - returns a string containing a printable representation of an object
- ▼ `bin()` - converts an integer to a binary string

```
print(bin(0))
print(bin(1))
print(bin(2))
print(bin(3))
print(bin(4))
print(bin(5))
print(bin(6))
print(bin(7))
print(bin(8))
print(bin(9))
print(bin(10))
print(bin(11))
print(bin(-1))
```

→

```
0b0
0b1
0b10
0b11
0b100
0b101
0b110
0b111
0b1000
0b1001
0b1010
0b1011
-0b1
```

- ▼ `bool()` - converts an argument to a Boolean value

```
print(bool(1))
print(bool(0))
print(bool(5))
print(bool(-0.4))
print(bool('lol'))
```

```
list = [0]
print(bool(list))
list1 = []
print(bool(list1))
```

→

True
False
True
True
True
True
False

- ▼ chr() - returns representation of character given by integer argument

```
print(chr(65))
print(chr(93))
```

→

A
J

Given an integer from 0 to 255, chr() returns str with the relevant character from ASCII table.

- ▼ complex() - returns a complex number constructed from arguments

complex() returns a number with two parts - real and imaginary (with j)

```
print(complex(2, -0.3))
print(complex(4))
```

→

(2-0.3j)
(4+0j)

- ▼ float() - returns a floating-point object constructed from a number or string

```
print(float(9))
print(float(0))
print(float("12.3"))
```

→

9.0

0.0

12.3

▼ `hex()` - converts an integer to a hexadecimal string

```
print(hex(5))
print(hex(277))
print(hex(-48))
print(hex(0))
```

→

0x5

0x115

-0x30

0x0

▼ `int()` - returns an integer object constructed from a number or a string

```
print(int('23'))
print(int(2.3))
print(int(-2.3))
```

→

23

2

-2

▼ `oct()` - converts an integer to an octal string

```
print(oct(8))
print(oct(-8))
```

→

0o10
-0o10

▼ **ord()** - returns integer representation of a character

The opposite to **chr()**

```
print(ord('A'))  
print(ord('a'))  
print(ord('z'))  
print(ord('Z'))
```

→

65
97
122
90

▼ **repr()** - returns a string containing a printable representation of an object

```
list = (0.1, -2)  
print(repr(1))  
print(repr('Lol'))  
print(repr(list))
```

→

1
'Lol'
(0.1, -2)

▼ **str()** - returns a string version of an object

```
list = (0.1, -2)  
dict = {"panda":23, 1:"bear"}  
print(str(1))  
print(repr(dict))  
print(str(list))  
print(type(str(dict)))
```

```
→  
1  
{'panda': 23, 1: 'bear'}  
(0.1, -2)  
<class 'str'>
```

- ▼ `type()` - returns the type of an object or creates a new type object

```
dict = {"panda":23, 1:"bear"}  
print(type(dict))
```

```
→  
<class 'dict'>
```

`type()` can also be used to create a class faster. These two methods create the same object:

```
>>>class Foo(object):  
...      bar =True
```

and

```
Foo = type('Foo', (), {'bar':True})
```

- ▼ Iterables and Iterators

- ▼ `all()` - returns True if all elements of an iterable are true

```
list = [0, 1, 2, "lol"]  
list1 = [1, 2, "lol"]  
print(all(list))  
print(all(list1))
```

```
→  
False  
True
```

- ▼ `any()` - returns True if any elements of an iterable are true

```
list = [0]
list1 = [0, 1, 2, "lol"]
print(any(list))
print(any(list1))
```

→

False

True

- ▼ `enumerate()` - returns a list of tuples containing indices and values from an iterable

```
list = [-1, 0, 1, 2, 3, 4]
for i in enumerate(list):
    print(i)
```

→

(0, -1)
(1, 0)
(2, 1)
(3, 2)
(4, 3)
(5, 4)

The first element of a tuple is index of an element of an iterable, and the second - its value.

- ▼ `filter()` - filters elements from an iterable

E.g.:

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
print(list(filter(lambda x: x[0].lower() in 'aieou', creature_names)))
```

→

['Odrey']

Read as:

I need to print a list

that is a result of function filter()

that is performed with usage of function lambda(),

where every element of an iterable is represented as x;

x[0] enables lambda() to go through every first letter of every word in the list creature_names,

lower - makes the first letters written in lowers case so that they were the same as our example ‘aieou’;

in the end I mention th elist where I need there operations to be done.

▼ iter() - returns an iterator object

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
iterated_creature_names = iter(creature_names)
print(next(iterated_creature_names))
print(next(iterated_creature_names))
print(next(iterated_creature_names))
print(next(iterated_creature_names))
```

→

Lola

Lilu

Jim

Odrey

The number of iterator object call outs must not exceed the number of objects in the iterable.

▼ len() - returns the length of an object

```
creature_names = ['Lola', 'Lilu', 'Jim', 'Odrey']
name = "Kiki"
print(len(creature_names))
print(len(name))
```

→

4

4

Doesn't work with int and float.

- ▼ map() - applies a function to every item of an iterable

```
list1 = [1, 2, 3, 4]
mapped_list = list(map(lambda x: x**5-1, list1))
print(mapped_list)
```

→

[4, 9, 14, 19]

```
aquarium_creatures = [
    {"name": "sammy", "species": "shark", "tank number": 11, "type": "fish"},
    {"name": "ashley", "species": "crab", "tank number": 25, "type": "shellfish"},
    {"name": "jo", "species": "guppy", "tank number": 18, "type": "fish"},
    {"name": "jackie", "species": "lobster", "tank number": 21, "type": "shellfish"},
    {"name": "charlie", "species": "clownfish", "tank number": 12, "type": "fish"},
    {"name": "olly", "species": "green turtle", "tank number": 34, "type": "turtle"}
]
def assign_to_tank(aquarium_creatures, new_tank_number):
    def apply(x):
        x["tank number"] = new_tank_number
        return x
    return map(apply, aquarium_creatures)
print(list(assign_to_tank(aquarium_creatures, 42)))
```

→

```
[{"name": "sammy", "species": "shark", "tank number": 42, "type": "fish"}, {"name": "ashley", "species": "crab", "tank number": 42, "type": "shellfish"}, {"name": "jo", "species": "guppy", "tank number": 42, "type": "fish"}, {"name": "jackie", "species": "lobster", "tank number": 42, "type": "shellfish"}, {"name": "charlie", "species": "clownfish", "tank number": 42, "type": "fish"}, {"name": "olly", "species": "green turtle", "tank number": 42, "type": "turtle"}]
```

```
base = [1, 2, 3]
powers = [1, 2, 3]
powered_numbers = list(map(pow, base, powers))
print(powered_numbers)
```

→

[1, 4, 27]

- ▼ next() - retrieves the next item from an iterator

```
numb = iter([1, 3, 5])
print(next(numb))
print(next(numb))
print(next(numb))
```

→

1
3
5

- ▼ range() - generates a range of integer values

```
for item in range(10):
    print(item)
```

→

0
1
2
3
4
5
6
7
8
9

▼ reversed() - returns a reverse iterator

```
numb = [1, 2, 3]
print(list(reversed(numb)))
```

→

[3, 2, 1]

▼ slice() - returns a slice object

```
quote = "Per aspera ad astra"
a = slice(4, 10)
b = slice(0, 3)
print(quote[a])
print(quote[b])
```

→

aspera

Per

▼ sorted() - returns a sorted list from an iterable

```
signs = ["Aries", "Taurus", "Gemini", "Cancer", "Leo", "Virgo", "Libra", \
         "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pieces"]
print(sorted(signs))
```

→

['Aquarius', 'Aries', 'Cancer', 'Capricorn', 'Gemini', 'Leo', 'Libra', 'Pieces', 'Sagittarius',
'Scorpio', 'Taurus', 'Virgo']

▼ zip() - creates an iterator that aggregates elements from iterables

```
signs = ["Aries", "Taurus", "Gemini", "Cancer", "Leo", "Virgo", "Libra", \
         "Scorpio", "Sagittarius", "Capricorn", "Aquarius", "Pieces"]
dates = ["(21st March - 20th April)", "(21st April - 21st May)", "(22nd May - 21st June)", "(22nd June - 22nd July)", "(23rd July - 23rd August)", "(24th August - 22nd September)", "(23rd September - 23rd October)", "(24th October - 22nd November)", "(23rd November - 21st December)", "(22nd December - 20th January)", "(21st January - 18th February)", "(19th February - 20th March)"]
```

```
for s, d in zip(signs, dates):
    print(s, d)
```

→

Aries (21st March – 20th April)
Taurus (21st April - 21st May)
Gemini (22nd May - 21st June)
Cancer (22nd June - 22nd July)
Leo (23rd July - 23rd August)
Virgo (24th August - 22nd September)
Libra (23rd September - 23rd October)
Scorpio (24th October - 22nd November)
Sagittarius (23rd November - 21st December)
Capricorn (22nd December - 20th January)
Aquarius (21st January - 18th February)
Pisces (19th February - 20th March)

▼ If/elif/else

▼ if...else...

```
French_vocab = {
    "food" : {
        "le fromage":"cheese",
        "le lait":"milk"
    },
    "animals" : {
        "le chat":"a cat",
        "l'elephant":"an elephant"
    },
    "time" : {
        "le soir": "tonight",
        "le matin": "this morning"
    }
}

def Antoshka():
    x = input('Введите слово: ')
    if x in French_vocab["animals"] or x in French_vocab["food"] or x in French_vocab["time"]:
        print("О, я знаю это слово! Но я забыл...")
    else:
        print("Это мы не проходили, это нам не задавали...")
```

```
Antoshka()
```

→

[input] le chat

[output] О, я знаю это слово! Но я забыл...

▼ elif

Stands in between:

```
French_vocab = {  
    "food" : {  
        "le fromage": "cheese",  
        "le lait": "milk"  
    },  
    "animals" : {  
        "le chat": "a cat",  
        "l'elephant": "an elephant"  
    },  
    "time" : {  
        "le soir": "tonight",  
        "le matin": "this morning"  
    }  
}  
  
def Antoshka():  
    x = input('Введите слово: ')  
    if x in French_vocab["food"] :  
        print("О, это что-то съедобное!")  
    elif x in French_vocab["animals"] or x in French_vocab["time"]:  
        print("Не знаю, что это, но точно не еда!")  
    else:  
        print("Это мы не проходили, это нам не задавали...")  
  
Antoshka()
```

→

[input] le chat

[output] Не знаю, что это, но точно не еда!

▼ Shorthand if

```
bag_items = ["sun_screen", "bottle_of_water", "magazine", "phone"]
x = input("Есть ли у Насти:")
if x in bag_items: print("Да, у Насти это есть!")
```

▼ Shorthand if...else...

This technique is known as **Ternary Operators**, or **Conditional Expressions**

```
bag_items = ["sun_screen", "bottle_of_water", "magazine", "phone"]
x = input("Есть ли у Насти:")
print("Да, у Насти это есть!") if x in bag_items else print("Такого нет!")
```

▼ Multiple conditions

```
x = int(input("Введите целое число: "))
print("Число больше нуля.") if x>0 else print("Число равно нулю.") if x==0 else print("Число меньше нуля.")
```

▼ Nested if

```
print('Добро пожаловать в нашу игру "Великолепный век!"')
x = input("Выберите пол персонажа: ").title()
if x == "Мужчина":
    y = int(input("Сколько вы задонатили в игру? Введите целое число: "))
    if y >= 1000:
        print("Поздравляем! Вы - султан! Вся власть в ваших руках!")
    else:
        print("Вы - слуга. Стоит задонатить еще, чтобы стать султаном!")
else:
    z = input("Напишите ваше главное качество: ").title()
    while z != "Красивая":
        s = input("Выберите другое качество: ").title()
        z //="Красивая"
    print("Подравляем! Вы в гареме!")
```

▼ pass statement

`if` statements cannot be empty, but if you for some reason have an `if` statement with no content, put in the `pass` statement to avoid getting an error.

```
a = 33  
b = 200  
if b > a:  
    pass
```

▼ While loop

▼ With the while loop we can execute a set of statements as long as a condition is true.

Note: remember to increment i, or else the loop will continue forever.

```
a = 0  
while a!=10:  
    print(a)  
    a = a+1
```

→

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

▼ break - stops the loop at the condition

```
a = 0  
while a!=10:  
    print(a)  
    if a == 5:  
        break  
    a = a+1
```

→

```
0  
1  
2  
3  
4  
5
```

- ▼ continue - stops prev. iteration and continues with the next one

```
i = 1  
while i < 6:  
    i *= 2  
    if i == 3:  
        continue  
    print(i)
```

→

```
2  
4  
8
```

- ▼ else

```
i = 1  
while i < 6:  
    print(i)  
    i *= 2  
else:  
    print("The end")
```

→

```
1  
2  
4  
The end
```

x//= “something”- another way to break the loop

▼ for loop

▼ loop

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    print(i)
```

→

a floor
windows
a door
a ceiling
walls

▼ break

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    print(i)
    if i == "a door":
        break
```

→

a floor
windows
a door

▼ continue

```
room = ["a floor", "windows", "a door", "a ceiling", "walls"]
for i in room:
    if i == "windows":
        continue
    print(i)
```

→

a floor
a door

a ceiling
walls

▼ range()

```
for x in range(10):
    print(x*3)
    x = x+1
```

→

0
3
6
9
12
15
18
21
24
27

▼ else

```
for x in range(5, 10, 2):
    print(x*3)
    x = x+1
else:
    print("Finish")
```

→

15
21
27
Finish

```
for x in range(5, 10):
    if x == 8: break
    print(x)
```

```
    else:  
        print("Finish")
```

→

5
6
7

▼ nested loops

▼ The "inner loop" will be executed one time for each iteration of the "outer loop":

```
girls = ["Sara", "Rachel", "Emily", "Lola"]  
boys = ["Sam", "Michael", "Jack", "Brandon"]  
  
def couples():  
    for x in girls:  
        for y in boys:  
            print(str(x), "+", str(y), "= a nice couple!")  
  
couples()
```

→

Sara + Sam = a nice couple!
Sara + Michael = a nice couple!
Sara + Jack = a nice couple!
Sara + Brandon = a nice couple!
Rachel + Sam = a nice couple!
Rachel + Michael = a nice couple!
Rachel + Jack = a nice couple!
Rachel + Brandon = a nice couple!
Emily + Sam = a nice couple!
Emily + Michael = a nice couple!
Emily + Jack = a nice couple!
Emily + Brandon = a nice couple!
Lola + Sam = a nice couple!
Lola + Michael = a nice couple!

Lola + Jack = a nice couple!
Lola + Brandon = a nice couple!

▼ pass

`for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.

```
for x in [0, 1, 2]:  
    pass
```

Composite Data Type

▼ `bytearray()` - creates and returns an object of the `bytearray` class (also known as “binary type”)

```
>>>b = bytearray(b'hello world!')  
>>>b  
bytearray(b'hello world!')  
>>>b[0]  
104  
>>>b[0] = b'h'  
Traceback (most recent call last):  
  File "", line 1, in  
    b[0] = b'h'  
TypeError: an integer is required  
>>>b[0] = 105  
>>>b  
bytearray(b'iello world!')  
>>>for i in range(len(b)):  
...     b[i] += i  
...>>>b  
bytearray(b'ifnos%}vzun,')
```

▼ `bytes()` - creates and returns a `bytes` object (similar to `bytearray` but immutable) (also known as “binary type”)

```
>>>b 'bytes'  
b'bytes'  
>>>'Байты'.encode('utf-8')  
b'\xd0\x91\xd0\xbd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'  
>>>bytes('bytes', encoding = 'utf-8')  
b'bytes'
```

```
>>>bytes([50, 100, 76, 72, 41])  
b'2dLH')
```

- ▼ dict() - creates a dict object (also known as “mapping type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",  
Taurus = "21st April - 21st May",  
Gemini = "22nd May - 21st June",  
Cancer = "22nd June - 22nd July",  
Leo = "23rd July - 23rd August",  
Virgo = "24th August - 22nd September",  
Libra = "23rd September - 23rd October",  
Scorpio = "24th October - 22nd November",  
Sagittarius = "23rd November - 21st December",  
Capricorn = "22nd December - 20th January",  
Aquarius = "21st January - 18th February",  
Pieces = "19th February - 20th March")  
print(zodiac_signs)  
print(zodiac_signs['Leo'])
```

→

```
{'Aries': '21st March – 20th April', 'Taurus': '21st April - 21st May', 'Gemini': '22nd May - 21st June', 'Cancer': '22nd June - 22nd July', 'Leo': '23rd July - 23rd August', 'Virgo': '24th August - 22nd September', 'Libra': '23rd September - 23rd October', 'Scorpio': '24th October - 22nd November', 'Sagittarius': '23rd November - 21st December', 'Capricorn': '22nd December - 20th January', 'Aquarius': '21st January - 18th February', 'Pieces': '19th February - 20th March'}  
23rd July - 23rd August
```

- ▼ frozenset() - creates a frozenset object (it's like set, but immutable)

```
b = frozenset('qwerty')  
print(b.add(1))
```

→

```
Traceback (most recent call last):  
File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 2, in <module>  
print(b.add(1))  
AttributeError: 'frozenset' object has no attribute 'add'
```

▼ `list()` - creates a list object (also known as “sequence type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(list(zodiac_signs))
```

→

```
['Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo', 'Libra', 'Scorpio', 'Sagittarius', 'Capricorn',
'Aquarius', 'Pieces']
```

▼ `object()` - creates a new featureless object

You cannot add new properties or methods to this object. This object is the base for all classes, it holds the built-in properties and methods which are default for all classes.

```
test = object()

print(type(test))
print(dir(test))
```

→

```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__',
 '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__']
```

In the program, we have used `type()` to get the type of the object. Similarly, we have used `dir()` to get all the attributes. These attributes (properties and methods) are common to instances of all Python classes.

▼ `set()` - creates a set object (also known as “set type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(set(zodiac_signs))
```

→

```
{'Scorpio', 'Gemini', 'Pieces', 'Virgo', 'Aquarius', 'Leo', 'Sagittarius', 'Aries', 'Libra', 'Taurus',
'Cancer', 'Capricorn'}
```

▼ `tuple()` - creates a tuple object (also known as “sequence type”)

```
zodiac_signs = dict(Aries = "21st March - 20th April",
Taurus = "21st April - 21st May",
Gemini = "22nd May - 21st June",
Cancer = "22nd June - 22nd July",
Leo = "23rd July - 23rd August",
Virgo = "24th August - 22nd September",
Libra = "23rd September - 23rd October",
Scorpio = "24th October - 22nd November",
Sagittarius = "23rd November - 21st December",
Capricorn = "22nd December - 20th January",
Aquarius = "21st January - 18th February",
Pieces = "19th February - 20th March")
print(tuple(zodiac_signs))
```

→

```
('Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo', 'Libra', 'Scorpio', 'Sagittarius', 'Capricorn',
'Aquarius', 'Pieces')
```

▼ `*memoryview` (also known as “binary type”) -

▼ `** range` (also known as “sequence type”) -

Arrays

▼ List (lis') - an ordered mutable collection of objects of various types (str and int cannot be in one list in Python3).

▼ Let's create a list:

- Way 1:

```
a = list('Morning')
print(a)
```

→

['M', 'o', 'r', 'n', 'i', 'n', 'g']

- Way 2:

```
a = [1, 2, 3]
print(a)
```

→

[1, 2, 3]

- Way 3:

```
list_generator = [c*3 for c in range(3)]
print(list_generator)
```

→

[0, 3, 6]

▼ List Methods

▼ list.append(x) - adds an elements to the end of a list

```
sample = [1, 2, 3]
sample.append(4)
```

```
print(sample)
sample.append(5)
print(sample)
```

→

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
```

- ▼ `list.extend(L)` - extends a list, adding elements from the list L to the end

```
sample = [1, 2, 3]
another_sample = [4, 5, 6]
sample.extend(another_sample)
print(sample)
print(another_sample)
```

→

```
[1, 2, 3, 4, 5, 6]
[4, 5, 6]
```

- ▼ `list.insert(i, x)` - pastes an x-element to the i-position in the list

```
sample = [1, 2, 3]
print(sample)
sample.insert(3, 4)
print(sample)
```

→

```
[1, 2, 3]
[1, 2, 3, 4]
```

- ▼ `list.remove(x)` - deletes the first x-element (if none - ValueError)

```
sample = [1, 2, 3, 2]
print(sample)
sample.remove(2)
print(sample)
```

→

[1, 2, 3, 2]

[1, 3, 2]

- ▼ list.pop(i) - deletes the element with i-index and returns the list. If list.pop() - deletes the last element

```
sample = [1, 2, 3, 2]
print(sample)
sample.pop(2)
print(sample)
```

→

[1, 2, 3, 2]

[1, 2, 2]

- ▼ list.index(x, [start [, end]]) - returns the index of the first x-element (searches from start to end)

```
sample = [1, 2, 2, 2, 3, 2]
print(sample)
print(sample.index(2))
```

→

[1, 2, 2, 2, 3, 2]

1

- ▼ list.count(x) - returns quantity of x-elements

```
sample = [1, 2, 2, 2, 3, 2]
print(sample)
print(sample.count(2))
```

→

[1, 2, 2, 2, 3, 2]

4

- ▼ list.sort(key=function/ None, reverse = True/ False) - sorts out the list according to the function given

```
sample_list = ['Rose', 'Lily', 'Sharon', 'Mio']
sample_list.sort()
print(sample_list)
sample_list.sort(reverse=True)
print(sample_list)

def sample_func (x):
    return len(x)
sample_list.sort(key=sample_func, reverse=True)
print(sample_list)
```

→

```
['Lily', 'Mio', 'Rose', 'Sharon']
['Sharon', 'Rose', 'Mio', 'Lily']
['Sharon', 'Rose', 'Lily', 'Mio']
```

▼ `list.reverse()` - returns a reversed list

```
sample_list = ['а р о з а у п а л а н а р у к у а з о р а']
sample_list.reverse()
print(sample_list)
sample_list.reverse()
print(sample_list)
```

→

```
['а р о з а у п а л а н а р у к у а з о р а']
['а р о з а у п а л а н а р у к у а з о р а']
```

▼ `list.copy()` - makes a shallow copy of a list

```
# mixed list
prime_numbers = [2, 3, 5]

# copying a list
numbers = prime_numbers.copy()

print('Copied List:', numbers)

# Output: Copied List: [2, 3, 5]
```

Another way to make a copy is to use the built-in method `list()`

```
thislist = ["apple", "banana", "cherry"]
```

```
mylist = list(thislist)
```

```
print(mylist)
```

▼ list.clear() - clears up a list

```
sample_list = ['а р о з а у п а л а н а р у к у а з о р а']  
sample_list.reverse()  
print(sample_list)  
sample_list.clear()  
print(sample_list)
```

→

```
['а р о з а у п а л а н а р у к у а з о р а']  
[]
```

List methods change the list itself, so we don't need to use a variable for the changed list.

▼ Looping a list

▼ Loop through a list

```
thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
    print(x)
```

→

```
apple  
banana  
cherry
```

▼ Loop by index

```
thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
    print(thislist[i])
```

→

```
apple  
banana  
cherry
```

▼ While Loop

```
thislist = [1, 2, 3, 4, 5, 6 ,7 ,8, 9]  
i = 0  
while i < len(thislist):  
    print(thislist[i])  
    i = i + 1
```

```
→  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

▼ Looping Using List Comprehension

```
thislist = ["apple", "banana", "cherry"]  
[print(x) for x in thislist]
```

```
→  
apple  
banana  
cherry
```

▼ List Comprehencion

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Syntax: newlist = [*expression* for *item* in *iterable* if *condition* == True]

The *condition* is like a filter that only accepts the items that evaluate to `True`.

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list.

▼ Without:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

→

`['apple', 'banana', 'mango']`

▼ With:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

→

`['apple', 'banana', 'mango']`

▼ More examples:

```
#Accept only numbers lower than 5:
newlist = [x for x in range(10) if x < 5]
print(newlist)
```

→

`[0, 1, 2, 3, 4]`

```

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x.upper() for x in fruits] #Set the values in the new list to upper case:

print(newlist)

newlist = ['hello' for x in fruits] #Set all values in the new list to 'hello':

print(newlist)

newlist = [x if x != "banana" else "orange" for x in fruits] #Return "orange" instead of "banana":

print(newlist)

```

→

```

['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
['hello', 'hello', 'hello', 'hello', 'hello']
['apple', 'orange', 'cherry', 'kiwi', 'mango']

```

▼ Nested

▼ Nested loops can also be used in a list comprehension expression. To obtain list of all combinations of items from two lists:

```

print([{x:y} for x in range(1, 15, 2) for y in range(1, 25, 2)])
def cons(word):
    result = [char for char in word if char not in ["a", "e", "i", "o", "u"]]
    print(result)
cons(word="siblings")
cons(word="country")

```

→

```

[{:1: 1}, {:1: 3}, {:1: 5}, {:1: 7}, {:1: 9}, {:1: 11}, {:1: 13}, {:1: 15}, {:1: 17}, {:1: 19}, {:1: 21}, {:1: 23}, {:3: 1}, {:3: 3}, {:3: 5}, {:3: 7}, {:3: 9}, {:3: 11}, {:3: 13}, {:3: 15}, {:3: 17}, {:3: 19}, {:3: 21}, {:3: 23}, {:5: 1}, {:5: 3}, {:5: 5}, {:5: 7}, {:5: 9}, {:5: 11}, {:5: 13}, {:5: 15}, {:5: 17}, {:5: 19}, {:5: 21}, {:5: 23}, {:7: 1}, {:7: 3}, {:7: 5}, {:7: 7}, {:7: 9}, {:7: 11}, {:7: 13}, {:7: 15}, {:7: 17}, {:7: 19}, {:7: 21}, {:7: 23}, {:9: 1}, {:9: 3}, {:9: 5}, {:9: 7}, {:9: 9}, {:9: 11}, {:9: 13}, {:9: 15}, {:9: 17},

```

```
{9: 19}, {9: 21}, {9: 23}, {11: 1}, {11: 3}, {11: 5}, {11: 7}, {11: 9}, {11: 11},  
{11: 13}, {11: 15}, {11: 17}, {11: 19}, {11: 21}, {11: 23}, {13: 1}, {13: 3},  
{13: 5}, {13: 7}, {13: 9}, {13: 11}, {13: 13}, {13: 15}, {13: 17}, {13: 19},  
{13: 21}, {13: 23}]  
['s', 'b', 'T', 'n', 'g', 's']  
['c', 'n', 't', 'r', 'y']
```

▼ Join Lists

▼ Concatenation

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
animals = ["koala", "bear", "flying fox", "rabbit", "turtle"]  
  
animals_and_their_food = fruits + animals  
print(animals_and_their_food)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'koala', 'bear', 'flying fox', 'rabbit', 'turtle']
```

▼ .append() + for

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
more_fruits = ["pear", "dragon fruit", "pineapple"]  
  
for x in more_fruits:  
    fruits.append(x)  
  
print(fruits)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'pear', 'dragon fruit', 'pineapple']
```

▼ .extend()

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
more_fruits = ["pear", "dragon fruit", "pineapple"]
```

```
fruits.extend(more_fruits)  
print(fruits)
```

→

```
['apple', 'banana', 'cherry', 'kiwi', 'mango', 'pear', 'dragon fruit', 'pineapple']
```

▼ Tuple (tuple) - an ordered immutable collection of objects (like list but immutable).

Immutability makes a tuple more secure than a list: it can't be changed neither intentionally (which is also sad) nor unintentionally (which is great).

Moreover, a tuple weights less, it is processed faster , and it can be used as a key in a dictionary.

You are allowed to **add tuples to tuples (or multiply a tuple)**, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple.

▼ Let's create a tuple

```
a = tuple('Hello, world!')  
b = ('hello', 'world')  
c = 'hello', 'world'  
print(a)  
print(b)  
print(c)  
print(type(a))  
print(type(b))  
print(type(c))
```

→

```
('H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')  
('hello', 'world')  
('hello', 'world')  
<class 'tuple'>  
<class 'tuple'>  
<class 'tuple'>
```

▼ Operations with tuples

```
b = ('hello', 'world')  
print(b[0])
```

```
b[0] = 'new'
```

→

hello

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3, in <module>

```
b[0] = 'new'
```

TypeError: 'tuple' object does not support item assignment

(we can take an element from a tuple, but not change or delete it; still we can delete a whole tuple)

```
b = ['hello', 'world']
print(tuple(b))
print(type(tuple(b)))
```

→

('hello', 'world')

<class 'tuple'>

```
b = ('hello', 'world')
print(list(b))
print(type(list(b)))
print(b)
print(type(b))
```

→

['hello', 'world']

<class 'list'>

('hello', 'world')

<class 'tuple'>

```
b = ('hello', 'world')
a = list(b)
a[0] = 'new'
print(tuple(a))
```

```
print(type(tuple(a)))
print(type(a))
```

→

```
('new', 'world')
<class 'tuple'>
<class 'list'>
```

▼ Tuple methods

- ▼ tuple.count() - returns a number of times a specified value occurs in a tuple

```
b = ('hello', 'world', 'hello')
print(b.count('hello'))
print(b.index('world'))
```

→

```
2
1
```

- ▼ tuple.index() - searches the tuple for a specified value and returns its position

```
b = ('hello', 'world', 'hello')
print(b.count('hello'))
print(b.index('world'))
```

→

```
2
1
```

▼ Unpacking a Tuple

In Python, we are also allowed to extract the values back into variables. This is called "unpacking":

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
```

```
print(green)
print(yellow)
print(red)
```

→

apple
banana
cherry

Note: *the number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.*

▼ Loop Tuples

▼ Loop through a tuple

```
butterflies = ("monarch", "pharaoh", "papillon")
for x in butterflies:
    print(x)
```

→

monarch
pharaoh
papillon

▼ Loop by index

```
butterflies = ("monarch", "pharaoh", "papillon")
for x in range(len(butterflies)):
    print(butterflies[x])
```

→

monarch
pharaoh
papillon

▼ While Loop

```
butterflies = ("monarch", "pharaoh", "papillon")
i = 0
while i < len(butterflies):
    print(butterflies[i])
    i = i+1
```

→

monarch
pharaoh
papillon

▼ Set (set) - an unordered collection of unique objects.

Set items are unchangeable, but you can remove items and add new items. Also, set items are unindexed, here's why there's no duplicate values in a set.

▼ Let's make a set

```
a = set()
print(a)
b = set("Morning!")
print(b)
c = {1, 2, 3, 4}
print(c)
d = {i for i in range(10)}
print(d)
e = {}
print(type(e))
```

→

{'o', 'n', 'M', 'i', 'g', '!', 'r'}
{1, 2, 3, 4}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
<class 'dict'>

▼ Access items

You cannot access items in a set by referring to an index or a key. But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.

▼ for loop

```
butterflies = {"monarch", "pharaoh", "papillon"}  
for x in butterflies:  
    print(x)
```

→

pharaoh
monarch
papillon

▼ in keyword

```
butterflies = {"monarch", "pharaoh", "papillon"}  
print("pharaoh" in butterflies)
```

→

True

▼ Loop set

```
colors = {'red', 'green', 'blue', 'pink'}  
for x in colors:  
    print(x)
```

→

blue
green
red
pink

Sets are handy when we need to get rid of repetitions:

```
a = ['cat', 'cat', 'mouse']  
print(a)  
print(set(a))
```

→

```
['cat', 'cat', 'mouse']
{'mouse', 'cat'}
```

frozenset - like set, but immutable

▼ Set Methods:

- ▼ a.union(b) - returns a set that is a merge of a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'fish', 'turtle', 'hamster'}
c = a.union(b)
print(c)
```

→

```
{'fish', 'hamster', 'cat', 'mouse', 'dog', 'turtle'}
```

- ▼ a.update(b) - adds elements from b to a (not only set, any iterable)

```
a = {'cat', 'dog', 'mouse'}
b = {'fish', 'turtle', 'hamster'}
a.update(b)
print(a)
```

→

```
{'dog', 'fish', 'mouse', 'cat', 'hamster', 'turtle'}
```

- ▼ a.intersection(b) - returns a set that is an intersection of a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.intersection(b)
print(c)
```

→

```
{'cat'}
```

- ▼ `a.intersection_update(b)` - leaves in the set a only those elements that are present in the set b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.intersection_update(b)
print(a)
```

→

{'cat'}

- ▼ `a.difference(b)` - returns the difference between a and b (those elements that are present in a but not in b)

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.difference(b)
print(c)
```

→

{'dog', 'mouse'}

- ▼ `a.difference_update(b)` - deletes from a elements from b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.difference_update(b)
print(a)
```

→

{'mouse', 'dog'}

- ▼ `a.symmetric_difference(b)` - returns symmetric difference between a and b (elements, present in a or b, but not in both)

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
c = a.symmetric_difference(b)
print(c)
```

→

```
{'hamster', 'dog', 'turtle', 'mouse'}
```

- ▼ `a.symmetric_difference_update(b)` - puts in a symmetric difference between a and b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
a.symmetric_difference_update(b)
print(a)
```

→

```
{'hamster', 'dog', 'mouse', 'turtle'}
```

- ▼ `a.issubset(b)` - returns True if a is a subset of b

```
a = {'cat', 'dog', 'mouse'}
b = {'cat', 'turtle', 'hamster'}
print(a.issubset(b))
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(c.issubset(d))
```

→

```
False
```

```
True
```

- ▼ `a.issuperset(b)` - returns True if b is a subset of a

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(d.issuperset(c))
```

→

```
True
```

- ▼ `a<b` - equal to $a \leq b$ and $a \neq b$

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(c<d)
```

→

True

- ▼ a>b - equal to a≥b and a≠b

```
c = {'cat'}
d = {'cat', 'turtle', 'hamster'}
print(d>c)
```

→

True

- ▼ set.add() - adds an element to the set

```
c = {'cat'}
c.add('turtle')
print(c)
```

→

{'turtle', 'cat'}

- ▼ set.clear() - removes all elements from the set

```
s = {'mouse', 'cat', 'house'}
print(s)
s.clear()
print(s)
```

→

{'mouse', 'house', 'cat'}
set()

- ▼ set.copy() - returns a copy of the set

```
s = {'mouse', 'cat', 'house'}
print(s.copy())
d = s.copy()
print(d)
```

→

```
{'mouse', 'house', 'cat'}
{'mouse', 'house', 'cat'}
```

▼ **set.discard()** - removes the specified element

```
s = {'mouse', 'cat', 'house'}
s.discard('mouse')
print(s)
```

→

```
{'house', 'cat'}
```

▼ **set.isdisjoint()** - returns whether two sets have an intersection (True if nothing in common)

```
s = {'mouse', 'cat', 'house'}
s1 = {'mouse', 'cat', 'castle'}
print(s1.isdisjoint(s))
s2 = {'house'}
s3 = {'mouse', 'cat', 'castle'}
print(s2.isdisjoint(s3))
```

→

False

True

▼ **set.pop()** - removes a random element from the set

```
s = {'mouse', 'cat', 'house'}
s.pop()
print(s)
s.pop()
print(s)
```

```
s.pop()  
print(s)
```

→

```
{'cat', 'mouse'}  
{'mouse'}  
set()
```

▼ **set.remove()** - removes a specified element from the set

```
s = {'mouse', 'cat', 'house'}  
s.remove('mouse')  
print(s)
```

→

```
{'cat', 'house'}
```

▼ **Dictionary** (dict) - unoredered collections of various values that have keys. One can get access to a value by using a key, and vice versa. Dict are also called “associative massives” or “hash-tables”. No duplicates (no two items with the same key).

▼ Let's create a dict

```
d = {} #using a literal  
print(d)  
d1 = {1:"Moon", 2:"Earth"} #using a literal  
print(d1)  
d2 = dict(race='elf', weapon='magic') #using dict function  
print(d2)  
d3 = dict([(1,'dollar'),(2,'ruble')]) #using a literal  
print(d3)  
d4 = dict.fromkeys(['b', 'a'], 100) #using fromkeys method  
print(d4)  
d5 = {x: x**3 for x in range(7)} #using a dictionary generator  
print(d5)
```

→

```
{}  
{1: 'Moon', 2: 'Earth'}
```

```
{'race': 'elf', 'weapon': 'magic'}  
{1: 'dollar', 2: 'ruble'}  
  
{'b': 100, 'a': 100}  
  
{0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216}
```

▼ Add items

```
thisdict = {  
    "brand": "Gucci",  
    "item": "bag",  
    "storage": 3  
}  
thisdict["color"] = "red"  
print(thisdict)
```

→

```
{"brand": 'Gucci', 'item': 'bag', 'storage': 3, 'color': 'red'}
```

▼ Remove items

Apart from `.pop()` and `.popitem()`, you can use `del` to delete a specific item from a dict:

```
thisdict = {  
    "brand": "Gucci",  
    "item": "bag",  
    "storage": 3  
}  
del thisdict["storage"]  
print(thisdict)
```

→

```
{"brand": 'Gucci', 'item': 'bag'}
```

`del` can also delete the whole dict

▼ Let's get a value and a key from a dict, and change values

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1[1])  
d1[2] = 'Mars'  
print(d1)
```

→

Moon

{1: 'Moon', 2: 'Mars'}

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1['Moon'])
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 2, in <module>

print(d1['Moon'])

KeyError: 'Moon'

▼ Loop a dict

▼ get keys

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",  
    "le chat": "a cat",  
    "le soir": "the evening"  
}  
  
for x in French_vocab:  
    print(x)
```

→

le fromage

le lait

le chat

le soir

▼ get values

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",
```

```
"le chat":"a cat",
"le soir":"the evening"
}

for x in French_vocab:
    print(French_vocab[x])
```

→

cheese
milk
a cat
the evening

▼ .values()

```
French_vocab = {
    "le fromage":"cheese",
    "le lait":"milk",
    "le chat":"a cat",
    "le soir":"the evening"
}

for x in French_vocab.values():
    print(x)
```

→

cheese
milk
a cat
the evening

▼ .keys()

```
French_vocab = {
    "le fromage":"cheese",
    "le lait":"milk",
    "le chat":"a cat",
    "le soir":"the evening"
}

for x in French_vocab.keys():
    print(x)
```

→

le fromage
le lait
le chat
le soir

▼ .items()

```
French_vocab = {  
    "le fromage": "cheese",  
    "le lait": "milk",  
    "le chat": "a cat",  
    "le soir": "the evening"  
}  
  
for x, y in French_vocab.items():  
    print(x, y)
```

→

le fromage cheese
le lait milk
le chat a cat
le soir the evening

▼ Copy a dict

Use .copy() or built-in func. dict()

▼ Nested dict

A dictionary can contain dictionaries, this is called nested dictionaries.

```
French_vocab = {  
    "food" : {  
        "le fromage": "cheese",  
        "le lait": "milk"  
    },  
    "animals" : {  
        "le chat": "a cat",  
        "l'elephant": "an elephant"  
    },  
    "time" : {  
        "le soir": "tonight",  
        "le matin": "this morning"
```

```

        }
    }

print(French_vocab)
print(French_vocab["time"])
for x in French_vocab:
    print(x)
for x in French_vocab["animals"].values():
    print(x)

```

→

```

{'food': {'le fromage': 'cheese', 'le lait': 'milk'}, 'animals': {'le chat': 'a cat', "l'elephant": "an elephant"}, 'time': {'le soir': 'tonight', 'le matin': 'this morning'}}}
{'le soir': 'tonight', 'le matin': 'this morning'}
food
animals
time
a cat
an elephant

```

▼ Dict Methods

▼ dict.clear() - clears up the dict

```

d1 = {1:"Moon", 2:"Earth"}
print(d1)
d1.clear()
print(d1)

```

→

```

{1: 'Moon', 2: 'Earth'}
{}

```

▼ dict.copy() - returns a copy of the dict

```

d1 = {1:"Moon", 2:"Earth"}
d2 = d1.copy()
print(d2)
print(d1)

```

→

```
{1: 'Moon', 2: 'Earth'}  
{1: 'Moon', 2: 'Earth'}
```

- ▼ dict.get(key, [default]) - returns the value of the key, if none - returns default (None)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.get(1))  
print(d1.get("Mars"))  
print(d1.get(3))
```

→

```
Moon  
None  
None
```

- ▼ dict.items() - returns key - value pairs that are in the dict

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.items())
```

→

```
dict_items([(1, 'Moon'), (2, 'Earth')])
```

- ▼ dict.keys() - returns keys

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.keys())
```

→

```
dict_keys([1, 2])
```

- ▼ dict.pop(key [, default]) - deletes the key and returns the value, if none - returns default (exception)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1)
```

```
print(d1.pop(1))
print(d1)
```

→

{1: 'Moon', 2: 'Earth'}

Moon

{2: 'Earth'}

- ▼ dict.popitem() - deletes and returns key-value pair, if {} - KeyError (no order)

```
d1 = {1:"Moon", 2:"Earth"}
print(d1)
print(d1.popitem())
print(d1)
print(d1.popitem())
print(d1)
print(d1.popitem())
print(d1)
```

→

{1: 'Moon', 2: 'Earth'}

(2, 'Earth')

{1: 'Moon'}

(1, 'Moon')

{}

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 7, in <module>

print(d1.popitem())

KeyError: 'popitem(): dictionary is empty'

- ▼ dict.setdefault(key[, default]) - returns the value of the key, if none - creates a key with default value (None)

```
d1 = {1:"Moon", 2:"Earth"}
print(d1.setdefault(1))
print(d1.setdefault(3))
```

→

Moon

None

- ▼ `dict.update([other])` - updates the dict by creating key-value pairs from [other]. Existing keys change. Returns None (not a new dict)

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1)  
d1.update({3:"Venus"})  
print(d1)  
d1.update({3:"Mars"})  
print(d1)
```

→

```
{1: 'Moon', 2: 'Earth'}  
{1: 'Moon', 2: 'Earth', 3: 'Venus'}  
{1: 'Moon', 2: 'Earth', 3: 'Mars'}
```

- ▼ `dict.values()` - returns values

```
d1 = {1:"Moon", 2:"Earth"}  
print(d1.values())
```

→

```
dict_values(['Moon', 'Earth'])
```

Classes and Objects

A **class** is like an object constructor, “a blueprint” for creating new objects (or subclasses). A class has features (attributes) and actions (methods) that objects of this class (or subclasses) inherit. A class can be created during runtime, and changed anytime.

An **object** - basically anything in Python (and everything has its class, hidden in the core of the language).

An object has:

- a state - represented by attributes, reflects the properties of an object;

- some behaviour - represented by methods of an object, reflects the response of an object to other objects;
- an identity - object is given a unique name which enables one object to interact with other objects.

When an object is created (declared), we say that a class has been instantiated.

▼ Let's create a class

```
class MyClass:  
    x = 5  
  
print(MyClass)
```

→

<class 'main.MyClass'>

▼ Let's create an object

```
class MyClass:  
    x = 5  
  
print(MyClass)  
  
p1 = MyClass()  
print(p1.x)
```

→

<class 'main.MyClass'>

5

▼ The `__init__()` Function

Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created

▼ Example

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
person1 = Person("Julie", 16)  
  
print(person1.name)  
print(person1.age)
```

→

Julie

16

The `__init__()` function is called automatically every time the class is being used to create a new object.

▼ Object Methods

Methods in objects are functions that belong to the object.

▼ Example

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def my_name_is(self):  
        print("Hello, my name is " + self.name + ".")  
  
    def im_age(self):  
        print("I'm " + str(self.age) + " years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()
```

▼ The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong to the class. It does not have to be named `self`, you can call it whatever you like, but it has to be the first parameter of any function in the class.

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(per):  
        print("Hello, my name is " + per.name + ".")  
  
    def im_age(per):  
        print("I'm " + str(per.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()
```

→

Hello, my name is Julie.

I'm 16 years old.

▼ Modify Object Properties

You can modify properties on objects like this:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(person1):  
        print("Hello, my name is " + person1.name + ".")  
  
    def im_age(person1):  
        print("I'm " + str(person1.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()  
  
person1.age = 17  
print((person1.age))
```

→

Hello, my name is Julie.

I'm 16 years old.

17

▼ Delete Object Properties

You can delete properties on objects by using the `del` keyword:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):  
        some_self.name = name  
        some_self.age = age  
  
    def my_name_is(person1):  
        print("Hello, my name is " + person1.name + ".")  
  
    def im_age(person1):  
        print("I'm " + str(person1.age) +" years old.")  
  
person1 = Person("Julie", 16)  
person1.my_name_is()  
person1.im_age()  
  
del person1.age  
print((person1.age))
```

→

Hello, my name is Julie.

I'm 16 years old.

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 17, in <module>

print((person1.age))

AttributeError: 'Person' object has no attribute 'age'

▼ Delete Objects

You can delete objects by using the `del` keyword:

▼ Example

```
class Person:  
    def __init__(some_self, name, age):
```

```

some_self.name = name
some_self.age = age

def my_name_is(person1):
    print("Hello, my name is " + person1.name + ".") 

def im_age(person1):
    print("I'm " + str(person1.age) +" years old.")

person1 = Person("Julie", 16)
person1.my_name_is()
person1.im_age()

del person1

print(person1)

```

→

Hello, my name is Julie.

I'm 16 years old.

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 18, in <module>

print((person1))

NameError: name 'person1' is not defined

▼ The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the `pass` statement to avoid getting an error.

▼ Example

```

class Person:
    pass
print(Person)

```

→

<class 'main.Person'>

```

class Alien:
    pass
print(Alien)

```

→

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 3

print(Alien)

^

IndentationError: expected an indented block

Class Methods:

- ▼ classmethod() - returns a class method for a function
- ▼ delattr() - deletes an attribute from an object
- ▼ getattr() - returns the value of a named attribute of an object
- ▼ hasattr() - returns True if an object has a given attribute
- ▼ isinstance() - determines whether an object is an instance of a given class
- ▼ issubclass() - determines whether an object is an instance of a given subclass
- ▼ property() - returns a property value of a class
- ▼ setattr() - sets the value of a named attribute of an object
- ▼ super() - returns a proxy object that delegates method calls to parent or sibling class

Input/ Output

- ▼ format() - converts a value to a formatted representation
- ▼ input() - reads input from the console
- ▼ open() - opens a file and returns a file object
- ▼ print() - prints to a text stream or the console

Variables, References, and Scope

- ▼ dir() - returns a list of names in current local scope or a list of object attributes
- ▼ globals() - returns a dictionary representing the current global symbol table
- ▼ id() - returns the identity of an object
- ▼ locals() - updates and returns a dictionary representing current local symbol table

- ▼ `vars()` - returns `_dict_` attribute for a module, class, or object

Miscellaneous

- ▼ `callable()` - returns `True` if object appears callable
- ▼ `compile()` - compiles source into a code or AST object
- ▼ `eval()` - evaluates a Python expression
- ▼ `exec()` - implements dynamic execution of Python code
- ▼ `hash()` - returns the hash value of an object
- ▼ `help()` - invokes the built-in help system
- ▼ `memoryview()` - returns a memory view object
- ▼ `staticmethod()` - returns a static method for a function
- ▼ `__import__()` - invoked by the `import` statement

Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class inherits from base class, and child class - from parent class.

- ▼ Example

```
class Person:  
    def __init__(self, fname, lname, gender):  
        self.firstname = fname  
        self.lastname = lname  
        self.gender = gender  
  
    def greeting(self):  
        print("Hello! My name is "+self.firstname+" "+self.lastname+".")  
  
class Student(Person):  
    def __init__(self, fname, lname, gender, faculty):  
        Person.__init__(self, fname, lname, gender)  
        self.university = "Harvard"  
        self.faculty = faculty  
  
    def myuniversity(self):  
        print("I study in "+self.university+".")
```

```

def myfaculty(self):
    print("I study at the "+self.faculty+" faculty.")

def accomodation(self):
    if self.gender == "male":
        print("I live in the male dormitory.")
    else:
        print("I live in the female dormitory.")

print("Here's a man who looks like being in his forties:\n")
y = Person("Finn", "Olleyson", "male")
y.greeting()
print()
print("Here's a girl who looks like being in her twenties:\n")
x = Student("Emma", "Tompson", "female", "Computer Science")
x.greeting()
x.myuniversity()
x.myfaculty()
x.accomodation()

```

→

Here's a man who looks like being in his forties:

Hello! My name is Finn Olleyson.

Here's a girl who looks like being in her twenties:

Hello! My name is Emma Tompson.

I study in Harvard.

I study at the Computer Science faculty.

I live in the female dormitory.

The child's `__init__()` function **overrides** the inheritance of the parent's `__init__()` function.

To keep the inheritance of the parent's `__init__()` function, add a call to the parent's `__init__()` function.

`super()` function makes the child class inherit all the methods and properties from its parent without naming the parent class.

Iterators

An iterator is an object that contains a countable number of values. It can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods `__iter__()` and `__next__()`.

Iterator vs Iterable

Lists, tuples, dictionaries, and sets (and str) are all iterable objects. They are iterable *containers* which you can get an iterator from.

- ▼ All these objects have a `iter()` method which is used to get an iterator:

```
tup = ("tiger", "lion", "panther")
myit = iter(tup)

print(next(myit))
print(next(myit))
print(next(myit))
```

→

tiger
lion
panther

- ▼ Loop

```
tup = ("tiger", "lion", "panther")
for x in tup:
    print(x)
```

→

tiger
lion
panther

- ▼ Class Iterator

To create an object/class as an iterator you have to implement the methods `__iter__()` and `__next__()` to your object.

```
class MyNumbers:
    def __iter__(self):
        self.a = 1
```

```

        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))

```

→

1
2
3
4
5

▼ StopIteration

To prevent the iteration to go on forever, we can use the `StopIteration` statement.

In the `__next__()` method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:

```

class MyNumbers:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        if self.a <= 20:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

```

```
for x in myiter:  
    print(x)
```

→

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

Sope

“Уровень вложенности” in Russian. It can be local or global.

Modules

Consider a module to be the same as a code library. A file containing definition of functions, classes, variables, constants or any other Python object, even some executable code you want to include in your application.

▼ Create a module

To create a module just save the code you want in a file with the file extension `.py`

▼ Use a module

Use the `import` statement (import a module). When using a function from a module, use the syntax: *module_name.function_name*

▼ Variables in a module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc).

▼ Naming and renaming

You can use any name.py. You can create an alias when you import a module, by using the `as` keyword (import some_module as sm).

▼ Built-in modules

To see the whole list print: `help('modules')`. Below there are some frequently used modules.

▼ os module

Has functions to perform many tasks of operating system.

▼ `mkdir()` - creates a new directory.

A new directory corresponding to path in string argument in the function will be created. If we open D drive in Windows explorer we should notice tempdir folder created.

```
>>> import os  
>>> os.mkdir("d:\\tempdir")
```

▼ `chdir()`- changes current working directory

```
>>> import os  
>>> os.chdir("d:\\temp")
```

▼ `getcwd()` - returns name of a current working directory

```
>>> os.getcwd()  
'd:\\temp'
```

Directory paths can also be relative. If current directory is set to D drive and then to temp without mentioning preceding path, then also current working directory will be changed to d:\temp.

```
>>> os.chdir("d:\\\\")
>>> os.getcwd()
'd:\\\\'
>>> os.chdir("temp")
>>> os.getcwd()
'd:\\\\temp'
```

In order to set current directory to parent directory use ".." as the argument to chdir() function.

```
>>> os.chdir("d:\\\\temp")
>>> os.getcwd()
'd:\\\\temp'
>>> os.chdir("..")
>>> os.getcwd()
'd:\\\\'
```

- ▼ rmdir() - removes a specific directory either with absolute or relative path (it shouldn't be current working directory, an it shouldn't be empty)

```
>>> os.chdir("tempdir")
>>> os.getcwd()
'd:\\\\tempdir'
>>> os.rmdir("d:\\\\temp")
PermissionError: [WinError 32] The process cannot access the file because it
is being used by another process: 'd:\\\\temp'
>>> os.chdir("..")
>>> os.rmdir("temp")
```

- ▼ listdir() - returns a list of all files in a directory

```
>>> os.listdir("c:\\\\Users")
['acer', 'All Users', 'Default', 'Default User', 'desktop.ini', 'Public']
```

- ▼ random module

Contains randomization functions.

Python uses a pseudo-random generator based upon Mersenne Twister algorithm that produces 53-bit precision floats. Functions in this module depend on pseudo-random number generator function `random()` which generates a random float number between 0.0 and 1.0.

- ▼ `random.random` - returns a random float number between 0.0 and 1.0 (no args).

```
>>> import random  
>>> random.random()  
0.755173688207591
```

- ▼ `random.randint()` - returns a random number between the given numbers

```
>>> random.randint(1,100)  
91
```

- ▼ `random.randrange()` - returns a random element from the range (start, stop, step args)

```
>>> random.randrange(0,101,10)  
40
```

- ▼ `random.choice()` - returns a randomly selected eleemnt from a sequence object (e.g. str, list, tuple).If empty - IndexError.

```
>>> import random  
>>> random.choice('computer')  
'o'
```

- ▼ `random.shuffle()` - randomly reorders elements in a list

```
>>> numbers=[12,23,45,67,65,43]  
>>> random.shuffle(numbers)  
>>> numbers  
[23, 12, 43, 65, 67, 45]
```

▼ math module

This module presents commonly required mathematical functions (trigonometric, representation, logarithmic, angle conversion functions).

In addition, two mathematical constants are also defined in this module (Pie, Euler's number, etc.).

▼ Trigonometric functions

▼ radians() - converts angle in degrees to radians

```
>>> math.radians(30)
0.5235987755982988
```

▼ degrees()- converts angle in radians in degrees

```
>>> math.degrees(math.pi/6)
29.99999999999996
```

▼ math.log() - returns natural logarithm of given number; natural logarithm is calculated on the base e

math.log10(): returns base-10 logarithm or standard logarithm of given number.

```
>>> math.log10(10)
1.0
```

▼ math.exp() - returns a float number after raising e to given number : exp(x) is equivalent to e**x

```
>>> math.e**10
22026.465794806703
```

▼ math.pow() - receives two args, raises first to the second, returns the result

```
>>> math.pow(4,4)  
256.0
```

▼ `math.sqrt()` - computes square root of given number

```
>>> math.sqrt(100)  
10.0
```

▼ Representation functions

▼ `math.ceil()` - approximates the given number to the smallest integer greater than or equal to the given float number

```
>>> math.ceil(4.5867)  
5
```

▼ `math.floor()` - returns the largest integer less than or equal to the given number

```
>>> math.floor(4.5687)  
4
```

▼ time module

▼ `time()` - returns current system time in ticks. The ticks is number of seconds elapsed after epoch time i.e. 12.00 am, January 1, 1970.

```
>>> time.time()  
1544348359.1183174
```

▼ `localtime()` - translates time in ticks in a time tuple notation.

```
>>> tk=time.time()  
>>> time.localtime(tk)  
time.struct_time(tm_year=2018, tm_mon=12, tm_mday=9, tm_hour=15, tm_min=11, tm_sec=25, tm_wday=6, tm_yday=343, tm_isdst=0)
```

- ▼ `asctime()` - returns a readable format of local time

```
>>> tk=time.time()  
>>> tp=time.localtime(tk)  
>>> time.asctime(tp)  
'Sun Dec  9 15:11:25 2018'
```

- ▼ `ctime()` - returns string representation of system's current time

```
>>> time.ctime()  
'Sun Dec  9 15:17:40 2018'
```

- ▼ `sleep()` - halts current program execution for a specified duration in seconds

```
>>> time.ctime()  
'Sun Dec  9 15:19:14 2018'  
>>> time.sleep(20)  
>>> time.ctime()  
'Sun Dec  9 15:19:34 2018'
```

▼ sys module

Provides functions and variables used to manipulate different parts of the Python runtime environment.

- ▼ `sys.argv` - returns the list of command line args passed to a Python script Item[0] - script name. Rest of the arguments are stored at subsequent indices.

Here is a Python script (`test.py`) consuming two arguments from command line.

```
import sys  
print ("My name is {}. I am {} years old".format(sys.argv[1], sys.argv[2]))
```

This script is executed from command line as follows:

```
C:\python37>python tmp.py Anil 23  
My name is Anil. I am 23 years old
```

▼ sys.exit - causes program to end and return to either Python console or command prompt. It is used to safely exit from program in case of exception.

▼ sys.maxsize - returns the largest integer a variable can take

```
>>> import sys  
>>> sys.maxsize  
9223372036854775807
```

▼ sys.path - an environment variable that returns search path for all Python modules

```
>>> sys.path  
['', 'C:\\\\python37\\\\Lib\\\\idlelib', 'C:\\\\python37\\\\python36.zip', 'C:\\\\python3  
7\\\\DLLs', 'C:\\\\python37\\\\lib', 'C:\\\\python37', 'C:\\\\Users\\\\acer\\\\AppData\\\\Ro  
aming\\\\Python\\\\Python37\\\\site-packages', 'C:\\\\python37\\\\lib\\\\site-packages']
```

▼ sys.stdin , sys.stdout , sys.stderr - file objects used by the interpreter for standard input, output and errors. stdin is used for all interactive input (Python shell). stdout is used for the output of print() and of input(). The interpreter's prompts and error messages go to stderr.

▼ sys.version - displays a string containing version number of current Python interpreter

▼ collections module

Provides alternatives to built-in container data types such as list, tuple and dict.

▼ namedtuple() - a factory function that returns object of a tuple subclass with named fields. Any valid Python identifier may be used for a field name except for names starting with an underscore.

```
collections.namedtuple(typename, field-list)
```

The typename parameter is the subclass of tuple. Its object has attributes mentioned in field list. These field attributes can be accessed by lookup as well as by its index.

Following statement declares a employee namedtuple having name, age and salary as fields

```
>>> import collections  
>>> employee=collections.namedtuple('employee', [name, age, salary])  
To create a new object of this namedtuple  
>>> e1=employee("Ravi", 251, 20000)
```

Values of the field can be accessible by attribute lookup

```
>>> e1.name  
'Ravi'
```

Or by index

```
>>> e1[0]  
'Ravi'
```

▼ `OrderedDict()` - Ordered dictionary is similar to a normal dictionary. However, normal dictionary the order of insertion of keys in it whereas ordered dictionary object remembers the same. The key-value pairs in normal dictionary object appear in arbitrary order.

```
>>> d1={}
>>> d1['A']=20
>>> d1['B']=30
>>> d1['C']=40
>>> d1['D']=50
```

We then traverse the dictionary by a for loop,

```
>>> for k,v in d1.items():
print (k,v)

A 20
B 30
D 50
C 40
```

But in case of `OrderedDict` object:

```
>>> import collections  
>>> d2=collections.OrderedDict()  
>>> d2['A']=20  
>>> d2['B']=30  
>>> d2['C']=40  
>>> d2['D']=50
```

Key-value pairs will appear in the order of their insertion.

```
>>> for k,v in d2.items():  
    print (k,v)  
A 20  
B 30  
C 40  
D 50
```

▼ deque() - A deque object supports append and pop operation from both ends of a list. It is more memory efficient than a normal list object because in a normal list, removing one of item causes all items to its right to be shifted towards left. Hence it is very slow.

```
>>> q=collections.deque([10,20,30,40])  
>>> q.appendleft(110)  
>>> q  
deque([110, 10, 20, 30, 40])  
>>> q.append(41)  
>>> q  
deque([0, 10, 20, 30, 40, 41])  
>>> q.pop()  
40  
>>> q  
deque([0, 10, 20, 30, 40])  
>>> q.popleft()  
110  
>>> q  
deque([10, 20, 30, 40])
```

▼ statistics module

▼ mean() - calculate arithmetic mean of numbers in a list

```
>>> import statistics  
>>> statistics.mean([2, 5, 6, 9])  
5.5
```

- ▼ median() - returns middle value of numeric data in a list. For odd items in list, it returns value at $(n+1)/2$ position. For even values, average of values at $n/2$ and $(n/2)+1$ positions is returned.

```
>>> import statistics  
>>> statistics.median([1, 2, 3, 8, 9])  
3  
>>> statistics.median([1, 2, 3, 7, 8, 9])  
5.0
```

- ▼ mode() - returns most repeated data point in the list

```
>>> import statistics  
>>> statistics.mode([2, 5, 3, 2, 8, 3, 9, 4, 2, 5, 6])  
2
```

- ▼ stdev() - calculates standard deviation on given sample in the form of list

```
>>> import statistics  
>>> statistics.stdev([1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5])  
1.3693063937629153
```

Some more modules ([TBA](#)):

- ▼ re module ([TBA](#))

Модуль для работы с регулярными выражениями

- ▼ match() - ищет последовательность в начале строки
- ▼ search() - ищет первое совпадение с шаблоном
- ▼ findall() - ищет все совпадения с шаблоном, возвращает результирующие строки в виде списка
- ▼ finditer() - ищет все совпадения с шаблоном, возвращает итератор

- ▼ `compile()` - компилирует регуляторное выражение (к этому объекту затем можно применять все перечисляемые функции)
 - ▼ `fullmatch()` - вся строка должна соответствовать описанному регулярному выражению
 - ▼ `re.sub()` - для замены в строках
 - ▼ `re.split()` - для разделения строки на части
- ▼ **threading module (TBA)**

Thread-based parallelism. This module constructs higher-level threading interfaces on top of the lower level `_tread` module.

- ▼ `active_count()` - return the number of Thread objects currently alive (the returned count is equal to the length of the list returned by `enumerate()`).
- ▼ `current_thread()` - return the current `Thread` object, corresponding to the caller's thread of control. If the caller's thread of control was not created through the `threading` module, a dummy thread object with limited functionality is returned.
- ▼ `excepthook()` - handle uncaught exception raised by `Thread.run()`.

The `args` argument has the following attributes:

- `exc_type`: Exception type.
 - `exc_value`: Exception value, can be `None`.
 - `exc_traceback`: Exception traceback, can be `None`.
 - `thread`: Thread which raised the exception, can be `None`.
- ▼ `__excepthook__` - holds the original value of `threading.excepthook()`. It is saved so that the original value can be restored in case they happen to get replaced with broken or alternative objects.
- ▼ `get_ident()` - return the ‘thread identifier’ of the current thread.
- ▼ `get_native_id()` - return the native integral Thread ID of the current thread assigned by the kernel
- ▼ `enumerate()` - return a list of all `Thread` objects currently active
- ▼ `main_thread()` - Return the main `Thread` object.

- ▼ `settrace(func)` - set a trace function for all threads started from the `threading` module
- ▼ `gettrace()` - get the trace function as set by `settrace()`
- ▼ `setprofile(func)` - Set a profile function for all threads started from the `threading` module
- ▼ `getprofile()` - get the profiler function as set by `setprofile()`
- ▼ `stack_size([size])` - return the thread stack size used when creating new threads
- ▼ `TIMEOUT_MAX()` - The maximum value allowed for the *timeout* parameter of blocking functions (`Lock.acquire()`, `RLock.acquire()`, `Condition.wait()`, etc.)
- ▼ tkinter module
- ▼ csv module
- ▼ pickle module
- ▼ socket module
- ▼ sqlite3 module
- ▼ json module
- ▼ itertools module (все они создают итераторы)
 - ▼ `count(start=, step=)` - returns an iterator of evenly spaced values (infinite iterator)

```
import itertools
x = itertools.count(start=10, step=5)
print(next(x))
print(next(x))
print(next(x))
print(next(x))
```

→

10, 15, 20, 25

```
import itertools
x = itertools.count(start=10, step=5)
for i in x:
    if i>35:
```

```
        break
    else:
        print(i, end=" ")
```

→

10 15 20 25 30 35

```
import itertools
l1 = [1, 2, 3, 4, 5]
l2 = zip(itertools.count(0, 1), l1)
print(list(l2))
```

→

[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)]

```
import itertools
l1 = map(lambda x: x**2, itertools.count(0, 1))
for i in l1:
    if i > 60:
        break
    else:
        print(i, end=" ")
```

→

0 1 4 9 16 25 36 49

- ▼ `cycle(obj)` - returns each element from given iterable and saves its copy, when iterating object ends, returns copy - this repetition forms an infinite loop (infinite iterator)

```
import itertools
l1 = [1, 2, 3]
l2 = itertools.cycle(l1)
count = 0
for i in l2:
    if count > 15:
        break
    else:
        print(i, end=" ")
    count += 1
```

→

1 2 3 1 2 3 1 2 3 1 2 3 1

```
import itertools
s = "Go! "
i = itertools.cycle(s)
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
```

→

G
o
!

G
o
!

- ▼ `repeat(obj, times)` - returns the object argument repeatedly (infinite iterator)

```
import itertools
s = "Go! "
i = itertools.repeat(s)
print(next(i))
print(next(i))
print(next(i))
```

→

Go!
Go!
Go!

```
import itertools
iterobj = itertools.repeat("Go!", times=10)
```

```
for i in iterobj:  
    print(i, end=" ")
```

→

Go! Go! Go! Go! Go! Go! Go! Go! Go!

```
import itertools  
powobj = map(pow, range(15), itertools.repeat(2))  
for i in powobj:  
    print(i, end=" ")
```

→

0 1 4 9 16 25 36 49 64 81 100 121 144 169 196

```
import itertools  
l = [1, 2, 3]  
obj = zip(itertools.repeat(5), l)  
print(list(obj))  
obj2 = zip(l, itertools.repeat(7))  
print(list(obj2))
```

→

[(5, 1), (5, 2), (5, 3)]

[(1, 7), (2, 7), (3, 7)]

```
import itertools  
l = ['Lolly', 'Molly', 'Polly']  
obj = zip(itertools.repeat("Hello"), l)  
print(next(obj))  
print(next(obj))  
print(next(obj))
```

→

('Hello', 'Lolly')

('Hello', 'Molly')

('Hello', 'Polly')

- ▼ `accumulate(iterable,func,*initial=None)` - applies a function to successive items in a list (finite iterator)

`functools.reduce()` возвращает только конечное накопленное значение для аналогичной функции.

```
import itertools
#using add and mul operator,so importing operator module
import operator
#using reduce(),so importing reduce() from functools module
from functools import reduce

l1=itertools.accumulate([1,2,3,4,5])
print(list(l1))
```

→

[1, 3, 6, 10, 15]

```
r1=reduce(operator.add,[1,2,3,4,5])
print (r1)
```

→

15

```
l2=itertools.accumulate([1,2,3,4,5],operator.add,initial=10)
print (list(l2))
```

→

[10, 11, 13, 16, 20, 25]

```
l3=itertools.accumulate([1,2,3,5,5],operator.mul)
print (list(l3))
```

→

[1, 2, 6, 30, 150]

```
r2=reduce(operator.mul,[1,2,3,4,5])  
print(r2)
```

→

120

```
l4=itertools.accumulate([2,4,6,3,1],max)  
print (list(l4))
```

→

[2, 4, 6, 6, 6]

```
r3=reduce(max,[2,4,6,3,1])  
print (r3)
```

→

6

```
l5=itertools.accumulate([2,4,6,3,1],min)  
print(list(l5))
```

→

[2, 2, 2, 2, 1]

```
r4=reduce(min,[2,4,6,3,1])  
print (r4)
```

→

1

▼ `chain()` - (finite iterator) создает итератор, который возвращает элемент из итерируемого объекта до тех пор, пока он не закончится, а потом переходит к

следующему. Он будет рассматривать последовательности, идущие друг за другом, как одну.(finite iterator)

```
import itertools

l1 = itertools.chain(["cat", "dog"], [5, 7, 9], "LOL")
print(list(l1))
```

→

['cat', 'dog', 5, 7, 9, 'L', 'O', 'L']

▼ `chain.from_iterable(iterable)` - берет один итерируемый объект в качестве входного аргумента и возвращает «склеенный» итерируемый объект, содержащий все элементы входного (finite iterator)

```
import itertools

l1 = itertools.chain.from_iterable(["kot", "kot", "kot"])
print((list(l1)))
```

→

['k', 'o', 't', 'k', 'o', 't', 'k', 'o', 't']

```
l1 = itertools.chain.from_iterable([1, 2, 3])
print((list(l1)))
```

→

Traceback (most recent call last):

File "C:\Users\belos\PycharmProjects\pythonProject\main.py", line 4, in <module>
print((list(l1)))

TypeError: 'int' object is not iterable

▼ `compress(data, selectors)` - фильтрует элементы *data*, возвращая только те, которые содержат соответствующий элемент в селекторах (*selectors*), стоящих в True. Прекращает выполнение, когда либо данные, либо селекторы закончились. (finite iterator)

```
import itertools

selectors1 = [True, False, True, False]
l1 = itertools.compress([1, 2, 3, 4], selectors1)
print(list(l1))
selectors2 = [False, False]
l2 = itertools.compress([1, 0], selectors2)
print(list(l2))
selectors3 = [True, True]
l3 = itertools.compress([0, 0, 0], selectors3)
print(list(l3))
```

→

[1, 3]

[]

[0, 0]

▼ `dropwhile(predicate, iterable)` - выбрасывает элементы из итерируемого объекта до тех пор, пока предикат (*predicate*) имеет значение `True`, а затем возвращает каждый элемент. Итератор не вернет выходных данных, пока предикат не получит значение `False`.(finite iterator)

```
import itertools

greater_three = itertools.dropwhile(lambda item: item<3, [1, 2, 3, 4])
print(list(greater_three))
```

→

[3, 4]

```
import itertools

return_if_first_odd = itertools.takewhile(lambda item: item%2!=0, [1, 3, 5,
4, 7])
print(list(return_if_first_odd))

return_if_first_even = itertools.takewhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are True to the predicate until it meets False, then iteration stops and function returns the iterable with the f
```

```

irst results

return_if_first_odd = itertools.dropwhile(lambda item: item%2!=0, [1, 3, 5,
4, 7, 6, 8, 9])
print(list(return_if_first_odd))

return_if_first_even = itertools.dropwhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are False to the predicate until
meets True, then iteration stops and function returns the iterable with the
first True result and the rest after it

```

→

[1, 3, 5]
[2, 2, 4, 8]
[4, 7, 6, 8, 9]
[1, 2, 4]

- ▼ `takewhile(predicate, iterable)` - возвращает элементы из итерируемого объекта до тех пор, пока предикат имеет значение True (finite iterator)

```

import itertools

greater_three = itertools.takewhile(lambda item: item<3, [1, 2, 3, 4])
print(list(greater_three))

```

→

[1, 2]

```

import itertools

return_if_first_odd = itertools.takewhile(lambda item: item%2!=0, [1, 3, 5,
4, 7])
print(list(return_if_first_odd))

return_if_first_even = itertools.takewhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are True to the predicate until
meets False, then iteration stops and function returns the iterable with the
first results

```

```

return_if_first_odd = itertools.dropwhile(lambda item: item%2!=0, [1, 3, 5,
4, 7, 6, 8, 9])
print(list(return_if_first_odd))

return_if_first_even = itertools.dropwhile(lambda x: x%2==0, [2, 2, 4, 8, 1,
2, 4])
print(list(return_if_first_even))

#takewhile - filters out those elements that are False to the predicate until
meets True, then iteration stops and function returns the iterable with the
first True result and the rest after it

```

→

- [1, 3, 5]
- [2, 2, 4, 8]
- [4, 7, 6, 8, 9]
- [1, 2, 4]

▼ `filterfalse()` - фильтрует элементы итерируемого объекта, возвращая только те, для которых предикат имеет значение `False`. Если предикат равен `None`, он возвращает элементы, которые стоят в значении `False`. (finite iterator)

```

import itertools

filter_odd = itertools.filterfalse(lambda item: item%2==0, [1, 2, 3, 5, 4, 7,
6, 8])
print(list(filter_odd))

filter_none = itertools.filterfalse(None, [0, 1, 2, 3, 4])
print(list(filter_none))

```

→

- [1, 3, 5, 7]
- [0]

▼ `zip_longest(iterables, fillvalue=None)` - агрегирует элементы из каждого итерируемого объекта. Если итераторы имеют неравномерную длину, то на место пропущенных значений ставится *fillvalue*. Итерация будет продолжаться до тех пор, пока не закончится самый длинный итерируемый объект. (finite iterator)

```

import itertools

zipped = itertools.zip_longest(["Gucci", "Chanel", "Prada"], ["in stock", "not in stock", "in stock", "not in stock", "in stock", "not in stock", "in stock", "not in stock"])
print(list(zipped))

zipped1 = itertools.zip_longest(["cat", "dog", "mouse", "hamster", "bunny"],
                               ["fed", "fed"], fillvalue="mb need to feed")
print(list(zipped1))

```

→

[('Gucci', 'in stock'), ('Chanel', 'not in stock'), ('Prada', 'in stock'), (None, 'not in stock'), (None, 'in stock'), (None, 'not in stock'), (None, 'in stock'), (None, 'not in stock')]
[('cat', 'fed'), ('dog', 'fed'), ('mouse', 'mb need to feed'), ('hamster', 'mb need to feed'), ('bunny', 'mb need to feed')]

- ▼ `starmap()` - вычисляет функцию, получая аргументы из итерируемого объекта.
 Используется вместо `map()`, когда параметры аргумента уже сгруппированы в кортежи в одном итерируемом объекте (данные были предварительно сжаты).
(finite iterator)

```

import itertools

a2=itertools.starmap(lambda x:x**2,[(1, ),(2, ),(3, )])
print (list(a2))

sq = itertools.starmap(pow, [(1, 2), (2, 2), (3, 2)])
print(list(sq))

```

→

[1, 4, 9]
[1, 4, 9]

- ▼ `islice()` - возвращает выбранные элементы из итерируемого объекта.
 Default `start` - 0, `step` - 1, `stop` - до конца итерируемого объекта. Если его нет, итератор остановится на определенной позиции. Значения `start`, `stop` и `step > 0`.
(finite iterator)

```
import itertools

s1 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], None)
print(list(s1))
s2 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 5)
print(list(s2))
s3 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 3, None, 2)
print(list(s3))
s3 = itertools.islice([1, 2, 3, 4, 5, 6, 7, 8], 2, 7, 2)
print(list(s3))
```

→

```
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5]
[4, 6, 8]
[3, 5, 7]
```

- ▼ `tee(iterable, n=2)` - возвращает n независимых итераторов из одного итерируемого объекта. (finite iterator)

```
import itertools

l1 = [1, 2, 3, 4, 5, 5, 6]
l2 = itertools.tee(l1, 2)
for i in l2:
    print(list(i))
```

→

```
[1, 2, 3, 4, 5, 5, 6]
[1, 2, 3, 4, 5, 5, 6]
```

- ▼ `groupby(iterable, key=None)` - возвращает последовательно ключи и группы из итерируемого объекта (finite iterator)

key – это функция, вычисляющая значение ключа для каждого элемента по умолчанию. Если ключ не указан или в значении `None`, то по умолчанию ключ является функцией идентификации, которая возвращает элемент без изменений.

```

import itertools

l1 = [("size", "XS"), ("size", "S"), ("size", "M"), ("quantity", 120), ("quantity", 134), ("size", "L")]
l2 = itertools.groupby(l1, key=lambda x:x[0])
for key, group in l2:
    result = {key: list(group)}
    print(result)
print()
l3 = [("gender", "female"), ("gender", "male"), ("age", 18), ("age", 21), ("age", 34)]
l4 = itertools.groupby(l3)
for key, group in l4:
    result = {key: list(group)}
    print(result)

```

→

```

{'size': [('size', 'XS'), ('size', 'S'), ('size', 'M')]}

{'quantity': [('quantity', 120), ('quantity', 134)]}

{'size': [('size', 'L')]}

{('gender', 'female'): [('gender', 'female')]}

{('gender', 'male'): [('gender', 'male')]}

{('age', 18): [('age', 18)]}

{('age', 21): [('age', 21)]}

{('age', 34): [('age', 34)]}

```

- ▼ `product(iterables, repeat)` - декартово произведение итерируемых объектов, подаваемых на вход.(combinatory iterator)

Декартово произведение - произведение множества X и множества Y – это множество, содержащее все упорядоченные пары (x, y) , в которых x принадлежит множеству X , а y принадлежит множеству Y .

Чтобы вычислить произведение итерируемого объекта умноженного самого на себя, нужно указать количество повторений с помощью опционального аргумента с ключевым словом `repeat`. Например, `product(A, repeat=4)` – тоже самое, что и `product(A, A, A, A)`.

```

import itertools

```

```

#Only one iterable is given
l1=itertools.product("ABCD")
print (list(l1))

#two iterables are given
l2=itertools.product("ABC", [1,2])
print (list(l2))

#one iterable and repeat is mentioned.
l3=itertools.product("xy", repeat=2)
print (list(l3))

l4=itertools.product("aa", repeat=2)
print (list(l4))

#More than two iterables is mentioned
l5=itertools.product([1,2], [3,4], [5,6])
print (list(l5))

```

→

```

[('A',), ('B',), ('C',), ('D',)]
[('A', 1), ('A', 2), ('B', 1), ('B', 2), ('C', 1), ('C', 2)]
[('x', 'x'), ('x', 'y'), ('y', 'x'), ('y', 'y')]
[('a', 'a'), ('a', 'a'), ('a', 'a'), ('a', 'a')]
[(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)]

```

- ▼ permutations() - возвращает последовательные **r** перестановок элементов в итерируемом объекте. (combinatory iterator)

Если **r** нет или **None**, то **r** = длине итерируемого объекта и генерирует все возможные полноценные перестановки. Кортежи перестановок выдаются в лексикографическом порядке в соответствии с порядком итерации входных данных. Т. о., если входные данные итерируемого объекта отсортированы, то комбинация кортежей будет выдаваться в отсортированном порядке.

Элементы рассматриваются как уникальные в зависимости от их позиции, а не от их значения. Т. о., если входные элементы уникальны, то в каждой перестановке не будет повторяющихся значений.

```

import itertools

l1=itertools.permutations("ABC")
print (list(l1))

l2=itertools.permutations([3,2,1])
print (list(l2))

#elements are treated as unique based on their position and not by their value.
l3=itertools.permutations([1,1])
print (list(l3))

l4=itertools.permutations(["ABC"])
print (list(l4))

#r value is mentioned as 2. It will return all different permutations in 2 values.
l5=itertools.permutations([1,2,3,4],2)
print (list(l5))

l4=itertools.product("aa", repeat=2)
print (list(l4))

#More than two iterables is mentioned
l5=itertools.product([1,2],[3,4],[5,6])
print (list(l5))

```

→

```

[('A', 'B', 'C'), ('A', 'C', 'B'), ('B', 'A', 'C'), ('B', 'C', 'A'), ('C', 'A', 'B'), ('C', 'B', 'A')]
[(3, 2, 1), (3, 1, 2), (2, 3, 1), (2, 1, 3), (1, 3, 2), (1, 2, 3)]
[(1, 1), (1, 1)]
[('ABC',)]
[(1, 2), (1, 3), (1, 4), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (3, 4), (4, 1), (4, 2), (4, 3)]
[('a', 'a'), ('a', 'a'), ('a', 'a'), ('a', 'a')]
[(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)]

```

▼ **combinations()** - возвращает подпоследовательности длины **r** из элементов итерируемого объекта, подаваемого на вход. (combinatory iterator)

Комбинация кортежей генерируется в лексикографическом порядке в соответствии с порядком элементов итерируемого объекта на входе. Таким образом, если входной итерируемый объект отсортирован, то комбинация кортежей будет генерироваться в отсортированном порядке.

Лексикографический порядок – способ упорядочивания слов в алфавитном порядке.

Элементы рассматриваются как уникальные в зависимости от их позиции, а не значения. Таким образом, если выходные элементы уникальны, то в каждой комбинации не будет повторяющихся значений.

```
import itertools

l5=itertools.combinations([1,2,3,4],2)
print (list(l5))
```

→

`[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]`

- ▼ `combinations_with_replacement()` - возвращает подпоследовательности длины `r` из элементов итерируемого объекта, подаваемого на вход, при этом отдельные элементы могут повторяться больше одного раза.

```
import itertools
l1=itertools.combinations("ABC",2)
print (list(l1))
l1=itertools.combinations_with_replacement("ABC",2)
print (list(l1))

l2=itertools.combinations([3,2,1],3)
print (list(l2))
l2=itertools.combinations_with_replacement([3,2,1],3)
print(list(l2))

#elements are treated as unique based on their position and not by their value.
l3=itertools.combinations([1,1],2)
print (list(l3))
l3=itertools.combinations_with_replacement([1,1],2)
print (list(l3))
```

```
#since list contains only one element, given r value is 2. So it returns empty list.  
l4=itertools.combinations(["ABC"],2)  
print (list(l4))  
#In combinations_with_replacement,it allows repeated element.  
l4=itertools.combinations_with_replacement(["ABC"],2)  
print (list(l4))
```

→

```
[('A', 'B'), ('A', 'C'), ('B', 'C')]  
[('A', 'A'), ('A', 'B'), ('A', 'C'), ('B', 'B'), ('B', 'C'), ('C', 'C')]  
[(3, 2, 1)]  
[(3, 3, 3), (3, 3, 2), (3, 3, 1), (3, 2, 2), (3, 2, 1), (3, 1, 1), (2, 2, 2), (2, 2, 1), (2, 1, 1),  
(1, 1, 1)]  
[(1, 1)]  
[(1, 1), (1, 1), (1, 1)]  
[]  
[('ABC', 'ABC')]
```

```
#r value is not mentioned. It will raise TypeError  
l5=itertools.combinations([1,2,3,4])  
print (list(l5))#Output:TypeError: combinations() missing required argument  
'r' (pos 2)  
l5=itertools.combinations_with_replacement([1,2,3,4])  
print (list(l5))#Output:TypeError: combinations_with_replacement() missing required argument 'r' (pos 2)
```

▼ dir()

- ▼ There is a built-in function to list all the function names (or variable names) in a module
 - the `dir()` function.

```
import platform  
  
x = dir(platform)  
print(x)
```

→

```
['_Processor', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES',
'builtins', 'cached', 'copyright', 'doc', 'file', 'loader', 'name', 'package', 'spec', 'version',
'_comparable_version', '_component_re', '_default_architecture', '_follow_symlinks',
'_get_machine_win32', '_ironpython26_sys_version_parser',
'_ironpython_sys_version_parser', '_java_getprop', '_libc_search', '_mac_ver_xml',
'_node', '_norm_version', '_platform', '_platform_cache', '_pypy_sys_version_parser',
'_sys_version', '_sys_version_cache', '_sys_version_parser', '_syscmd_file',
'_syscmd_ver', '_uname_cache', '_unknown_as_blank', '_ver_output', '_ver_stages',
'architecture', 'collections', 'functools', 'itertools', 'java_ver', 'libc_ver', 'mac_ver',
'machine', 'node', 'os', 'platform', 'processor', 'python_branch', 'python_build',
'python_compiler', 'python_implementation', 'python_revision', 'python_version',
'python_version_tuple', 're', 'release', 'subprocess', 'sys', 'system', 'system_alias', 'uname',
'uname_result', 'version', 'win32_edition', 'win32_is_iot', 'win32_ver']
```

▼ Import from a module →

You can choose to import only parts from a module, by using the `from` keyword (from sm import sm_person).

When importing using the `from` keyword, do not use the module name when referring to elements in the module. Example: `person1["age"]`, not `mymodule.person1["age"]`

▼ Custom modules

▼ Attributes

▼ __name__

When Python is running in interactive mode or as a script, its `__name__` is `_main_`. It is the name of scope in which top level is being executed. However, for imported module this attribute is set to name of the Python script. (excluding the extension .py).

▼ __doc__

This attribute returns the docstring of module. Just as in function, Documentation string (docstring) is a string literal in first line written in module code.

▼ __file__

This attribute will returns the name and path of the module file.

▼ __dict__

This attribute returns a dictionary object of all attributes, functions and other definitions and their respective values.

▼ Module search

How does Python interpreter find a module when a script requests it to be imported? Python follows following method to locate a module:

- Whether it is present in current working directory?
- If not found there, directories in PYTHONPATH environment variable are searched.
- Otherwise, it searches the installation dependent default directory.

▼ Reloading a module

Python loads any module only once even if import statement is issued repeatedly.

```
"docstring of messages module"
def welcome(name):
    print ("Hi {}. Welcome to Python Tutorial".format(name))
    return
def bye(name):
    print ("Good Bye {}. See you again".format(name))
welcome("world")
bye("world")
```

Let us import messages module from Python prompt. The calls to functions in the code also get executed. However, repeating import statement doesn't execute them again.

```
>>> import messages
Hi world. Welcome to Python Tutorial
Good Bye world. See you again
>>> import messages
>>>
```

Now suppose we need to modify the SayHello() function before executing it again. Updated function is :

```
"docstring of messages module"
def welcome(name):
    print ("Hi {}. Welcome to Python Tutorial".format(name))
    return
def bye(name):
```

```
    print ("Good Bye {}. See you again".format(name))
welcome("Guest")
bye("Guest")
```

This change will be effected only if current interpreter session is closed and re-launched. If such changes are to be done frequently, to close and restart Python is cumbersome. In order to call load the module without terminating interpreter session, use reload() function from imp module.

```
>>> import imp
>>> imp.reload(messages)
Hi Guest. Welcome to Python Tutorial
Good Bye Guest. See you again
<module 'messages' from 'C:\\python37\\messages.py'>
```

Decorators

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-decorators>

CGI in Python

Common Getaway Interface - стандарт для внешних шлюзовых программ для взаимодействия с информационными серверами (пр.: HTTP-серверы). Это набор соглашений, к-й должен соблюдаться web-серверами при выполнении ими различных web-приложений.

Переменные окружения

- ▶ QUERY_STRING – храниться значения
- ▶ REQUEST_METHOD – храниться метод передачи данных get или post
- ▶ CONTENT_TYPE – хранит тип передачи данных
Для get обычно используется "application/x-www-form-urlencoded"
Для post обычно используется "multipart/form-data"
- ▶ HTTP_HOST – хранит имя хоста с портом
- ▶ SERVER_NAME - хранит имя хоста
- ▶ HTTP_USER_AGENT – хранит сведения о клиенте "Mozilla/5.0 (Windows NT 6.3; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0"
- ▶ HTTP_REFERER – формируется автоматически браузером и хранит url страницы с которой пришёл запрос

▶ CONTENT_LENGTH – хранит строку, являющуюся десятичным представлением длины данных в байтах. Присутствует только в методе POST, в GET отсутствует

▶ HTTP_COOKIE – хранит все cookies в URL-кодировке

▶ HTTP_ACCEPT – хранит перечисления типов данных документа который принимает браузер например: "text/html, text/plain, image/gif, image/jpeg", но все новые браузеры передают "*" – это значит что принимают все типы.

Подробнее: <https://andreyex.ru/yazyk-programmirovaniya-python/uchebnik-po-python-3/python-3-programmirovaniye-v-python-cgi/>

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-cgi>

Django

Документация: <https://docs.djangoproject.com/en/4.0/ref/contrib/messages/>

▼ Методика MTV (или MVC)

Описывает общий дизайн БД ориентированных веб-приложений Django.

Django поощряет свободное связывание и строгое разделение частей приложения. Поэтому можно легко вносить изменения в одну конкретную часть приложения без ущерба для остальных частей. Так, мы используем тот же принцип разделения, как если бы отделяли бизнес-логику от логики отображения с помощью шаблонной системы.

Эти три вещи вместе — логика доступа к данным, бизнес-логика и логика отображения — составляют концепцию, которую называют шаблоном *Модель-Представление-Управление*

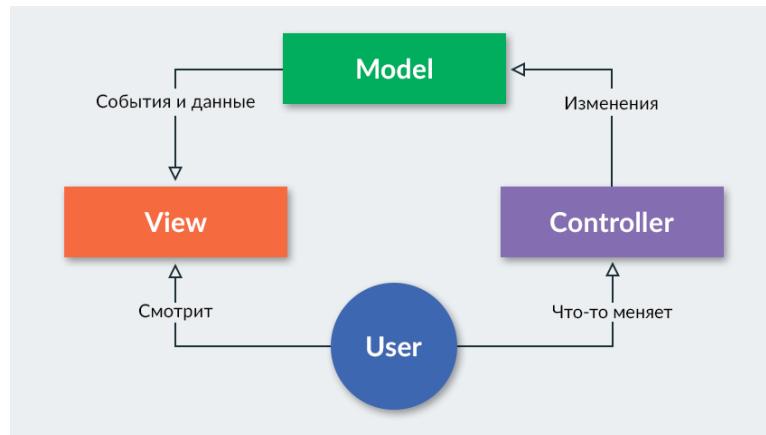
(*Model-View-Controller*, MVC) архитектуры программного обеспечения. В этой концепции термин «Модель» относится к логике доступа к данным; термин «Представление» относится к той части системы, которая определяет, что показать и как; а термин «Управление» относится к той части системы, которая определяет какое представление надо использовать, в зависимости от пользовательского ввода, по необходимости получая доступ к модели.

Django следует модели MVC достаточно близко, т.е., может быть назван MVC совместимой средой разработки. Вот примерно как M, V и C используются в Django:

- *M*, доступ к данным, обрабатывается слоем работы с базой данных, который описан в этой главе.
- *V*, эта часть, которая определяет какие данные получать и как их отображать, обрабатывается представлениями и шаблонами.
- *C*, эта часть, которая выбирает представление в зависимости от пользовательского ввода, обрабатывается самой средой разработки, следуя созданной вами схемой URL, и вызывает соответствующую функцию Python для указанного URL.

Так как «С» обрабатывается средой разработки и всё интересное в Django происходит в моделях, шаблонах и представлениях, на Django ссылаются как на *MTV-ориентированную среду разработки*. В MTV-подходе к разработке:

- *M* определено для «Модели» (Model), слоя доступа к данным. Этот слой знает всё о данных: как получить к ним доступ, как проверить их, как с ними работать и как данные связаны между собой.
- *T* определено для «Шаблона» (Template), слоя представления данных. Этот слой принимает решения относительно представления данных: как и что должно отображаться на странице или в другом типе документа.
- *V* определено для «Представления» (View), слоя бизнес-логики. Этот слой содержит логику, как получать доступ к моделям и применять соответствующий шаблон. Вы можете рассматривать его как мост между моделями и шаблонами.



REST Framework

CORS headers

▼ Протоколы прикладного уровня

- [9P](#)
- [BitTorrent](#)
- [BOOTP](#)
- [DNS](#)
- [FTP](#)
- [HTTP](#)

- NFS
- POP, POP3
- IMAP
- RTP
- SMTP
- SNMP
- SPDY
- Telnet
- SSH
- X.400
- X.500
- RDP
- Matrix
- SFTP

Компьютерные сети:

- по типу коммутации: комм. каналов (по каналу, пр.: тел. сети) и комм. пакетов (каждый пакет по своему пути, пр.: комп. сети)
- по технологии передача: широковещательные сети (сразу всем устр.) и “точка-точка”(от одного другому, мб посредством третьего и т.д.)
- протяженность
(персональная → локальная → муниципальная → глобальная → объединение сетей)

Топология КС - способ объединения комп. в сеть (еще наз. “граф”)

Вершины графа - узлы сети (комп. и сет. устр), ребра - связи между узлами (физ. и инф.)

- полносвязная Т - каждый соед. с каждым
- ячеистая Т - как полносвязная, но без некоторых соед.
- звезда - все подсоединенены к центральному устр-ву (коммутатор, ви-фи, маршрутизатор и т.д.), через к-е и осущ. передача данных

- кольцо - передача по кругу через соседей
- дерево - комп. и ЦУ обр-т дерево (ЦУ → др. ЦУ → комп.)
- общая шина - все комп. подкл к общ. сети передачи (пр. медный кабель), как к каналу
- смешанная - (на практике) - пр.: ЦУ в кольце, к ним подкл-ы комп-ы по звезде, а ост-е - по дереву через доп. ЦУ.

Физич. Т - соед. устр-в в сети. Логич. Т - правила расп-я сигналов в сети. Могут различаться: Ethernet - физ-ки звезда, лог-ки шина. Коммутир. Ethernet - ф-ки звезда, л-ки полносвяз. WiFi - физ. - нет, лог. - шина.

Стандарты сетей:

- Эталонная модель взаимодействия открытых сетей (принята OSI, м-н орг. по стандартизации)
- Технологии передачи данных (институт инж. по эл-ке и электротехнике, IEEE)
- Протоколы интернет (прин. Совер по арх. интернета, IAB)
- стандарты Web (консорциум W3C)

Для решения сложностей в организации масштабируемой КС был создан метод декомпозиции задач “уровни”. У кажд. ур. своя задача (или набор связ. задач).

Сервис - опис-т фу-ии ур-я.	Интерфейс - набор примитив. опер., к-е ниж. ур. предост. верх.	Протокол - прав. и соглаш. для связи ур. N 1 комп. с ур. N 2 комп.
--------------------------------	--	---

Арх. сети - набор ур. и протоколов (интерфейсы не входят). Стек протоколов собран иерархически.

Модель OSI

Хорошая теор. детализация, но на практ не прим-ся. Open Systems Interconnection, Модель взаимодействия открытых систем. Октр. сист. - в соотв. с открытыми спецификациями.

7. Ур. приложения (сообщение) - он же прикладной - включает все сервисы, к-ми может пользоваться пользователь (гипертекстовые web-страницы, соц. сети, видео и аудио связь, эл. почта, доступ к разделяемым файлам и т.д.).

Здесь располагаются сетевые службы, позволяющие пользователю взаимодействовать с машиной: Telnet, LPD, TFTP, NFS, DNS, DHCP, SNMP, X Window.

- Протокол HTTP (Hyper-Text Transfer Protocol) - протокол передачи гипертекста, основа www.

URL (Uniform Resource Locator) - уникальное положение ресурса. HTTP работает в режиме запрос-ответ.

Постоянное TCP соединение служит для того, чтобы по протоколу HTTP передавалось не по 1 файлу за соединение, а сразу все. (keep-alive или persistent connection).

В HTTP 1.0 нужно добавить строчку Connection: keep-alive, а в HTTP 1.1 все соединения по умолчанию постоянные (чтобы разорвать, нужно добавить заголовок Connection: close)

Минусы: клиент открыл соединение и не использует его - ресурсы недоступны другим, плохо для высоконагруженных серверов. Решение: автоотключение через таймаут 5-15 сек., или еще конвейреная обработка (pipelining) (несколько запросов → несколько ответов, недостаток - сохранение порядка, решена в HTTP2, где есть нумерация запросов), или еще вариант - несколько HTTP соединений (каждое м.б. постоянным или использовать конвейерную обработку). Последний вариант сейчас чаще всего используется.

Кэширование сокращает время загрузки страницы, но требует место на лок. диске комп-а. Может кэшировать отдельные ресурсы, к-е редко меняются.

Заголовок Expires помогает понять, можно ли взять страницу из кэша или ее нужно обновить. Если в браузере такого нет, используются эвристики (Last-Modified). Другой способ (нивелирующий ошибки предыдущих) - GET Conditional (при наличии Last-Modified): были ли изменения? В совр. версиях HTTP также используется ETag (entity tag) в запросах GET с условием.

Заголовок Cache-Control служит для управления кэшем.

Альтернатива хранению на лок. диске комп-а - прокси-сервер с разделяемым кэшем.

Есть еще обратный прокси, к-й устанавливается не со стороны клиента, а со стороны вэб-серверов.

Порядок следования HTTP сообщение:

- HTTP-метод
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (<http://>), домена (здесь developer.mozilla.org), или TCP порта (здесь 80)
- Версию HTTP-протокола

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера
- Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс

Методы HTTP

GET – запрос Web-страницы
POST – передача данных на Web-сервер
HEAD – запрос заголовка страницы
PUT – помещение страницы на Web-сервер
DELETE – удаление страницы с Web-сервера
TRACE – трассировка страницы
OPTIONS – запрос поддерживаемых методов HTTP для ресурса
CONNECT – подключение к Web-серверу через прокси

Статусы HTTP

1XX – информация
2XX – успешное выполнение (200 OK)
3XX – перенаправление (301 – постоянное перемещение, 307 – временное перенаправление)
4XX – Ошибка на стороне клиента (403 – доступ запрещен, 404 – страница не найдена)
5XX – Ошибка сервера (500 – внутренняя ошибка сервера)

Структура пакета HTTP

→ Запрос/статус ответа

- GET /courses/networks
- 200 OK

Заголовки (не обязательно)

- Host: www.asozykin.ru (обязательно в HTTP 1.1)
- Content-Type: text/html; charset=UTF-8
- Content-Length: 5161

Тело сообщения (не обязательно)

- Страница HTML
- Параметры, введенные пользователем

Пример запроса HTTP

Подключение по TCP к серверу www.asozykin.ru,
порт 80

```
-----  
GET /courses/networks HTTP/1.1  
Host: www.asozykin.ru
```

Пример ответа HTTP

```
HTTP/1.1 200 OK  
Server: nginx  
Content-Type: text/html; charset=UTF-8  
Content-Length: 5161
```

```
<html lang="ru-RU">  
<head>  
...  
</html>
```

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь `developer.mozilla.org`), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь `developer.mozilla.org`), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх частей (рис. 1), которые передаются в следующем порядке:

- HTTP-метод, обычно глагол подобно GET, POST или существительное, как OPTIONS или HEAD, определяющее операцию, которую клиент хочет выполнить. Обычно, клиент хочет получить ресурс (используя GET) или передать значения HTML-формы (используя POST), хотя другие операции могут быть необходимы в других случаях.
- Путь к ресурсу: URL ресурсы лишены элементов, которые очевидны из контекста, например без протокола (`http://`), домена (здесь developer.mozilla.org), или TCP порта (здесь 80).
- Версию HTTP-протокола.
- Заголовки (опционально), предоставляющие дополнительную информацию для сервера.

Или тело, для некоторых методов, таких как POST, которое содержит отправленный ресурс.

- Протокол SMTP (Simple Mail Transfer Protocol) - простой протокол передачи почты (посл. версия - ESMTP, extended). Схему использования протокола при передаче почты - см. ниже.

Теоретически SMTP может использовать любой транспортный протокол (TCP, UDP, др.). Порты: 25 для передачи почты между серверами, 587 для приема почты от клиентов. На практике: протокол TCP, порт 25.

Протокол SMTP входит в конверт письма, а заголовки и содержимое письма включают RFC 2822.

Протокол работает в формате текста в режиме запрос-ответ.

Минусы протокола: незащищенность данных, спам.

Команды SMTP

Команда	Назначение	Пример
HELO	Установка соединения	HELO example.com
MAIL	Адрес отправителя	MAIL FROM: sender@example.com
RCPT	Адрес получателя	RCPT TO: recipient@mail.ru
DATA	Передача письма	DATA
QUIT	Выход	QUIT

Ответы SMTP

Код	Назначение	Пример
220	Подключение к серверу успешно	220 smtp.example.com ESMTP Postfix
250	Успешное выполнение предыдущей команды	250 Hello example.com 250 Ok
354	Начало передачи письма	354 End data with <CR><LF>.<CR><LF>
502	Команда не реализована	502 5.5.2 Error: command not recognized
503	Неправильная последовательность команд	503 5.5.1 Error: need MAIL command
221	Закрытие соединения	221 2.0.0 So long, and thanks for all the fish

Заголовки письма

Заголовок	Назначение
From:	Отправитель (имя и адрес)
To:	Получатель
CC:	Получатель копии письма
BCC:	Получатель копии, адрес которого не должен быть показан
Reply-To:	Адрес для ответа
Subject:	Тема письма
Date:	Дата отправки письма

- Протокол IMAP (Internet Message Access Protocol) - протокол доступа к эл. почте. Письма хранятся на почтовом сервере. Клиенты подключаются к серверу и загружают письма только после запроса пользователя. Сервер может выполнять сложные операции с письмами.

Преимущества: одновременно могут работать несколько клиентов, все клиенты видят одно и то же состояние почтового ящика.

Недостатки: более сложный протокол, ограниченное место для почтового ящика на сервере.

IMAP использует TCP, порт 143.

IMAP позволяет использовать несколько почтовых ящиков и папок (хранятся на сервере, можно перемещать и образовывать иерархию). Также есть флаги.

Состояния сеанса: клиент не аутентифицирован → клиент аутентифицирован → папка выбрана → выход.

Работает в текстовом режиме, взаимодействие запрос-ответ. Позволяет выполнять несколько команд одновременно (есть теги команд).

Статусы: OK, NO (ошибка), BAD (неправильная команда).

- Протокол POP3 (Post Office Protocol) - протокол почтового отделения. Работает по принципу загрузить и удалить. Ящик на сервере - временное хранилище, сообщения переписываются на почтовый клиент, после чего удаляются с сервера.

Плюсы: простота, письма доступны при отсутствии подключения к сети. Минусы: только один клиент, единое хранилище писем (нет папок, фильтров и т.д.).

Состояния сеанса: авторизация, транзакция, обновление.

Протокол работает в текстовом режиме, взаимодействие запрос-ответ.

Ответы протокола: + OK или - ERR.

Команды POP3

Команда	Назначение	Пример
USER	Указать имя пользователя	USER asozykin
PASS	Указать пароль	PASS 1234qwer
STAT	Количество писем на сервере	STAT
LIST	Передача информации о сообщениях	LIST 2
RETR	Передать сообщение на клиент	RETR 1
TOP	Передать на клиент заголовок сообщения	TOP 2 10
DELE	Пометить сообщение на удаление	DELE 1
QUIT	Закрытие транзакции, удаление сообщений и отключение	QUIT

- **Протокол DNS** (Domain Name System) - система доменных имен. Позволяет использовать вместо неудобных IP понятные для пользователя доменные имена. Также позволяет менять сетевую инфраструктуру (при переходе на другой IP домен менять не надо). Один домен может обслуживать несколько серверов. Узнать IP по домену можно с помощью утилиты nslookup. Для Linux это утилиты host и dig.

Особенности DNS:

- распределенная (децентрализованная) система (нет единого сервера, на котором описываются имена хостов)
- делегирование ответственности (пространство имен разделено на отдельные части - домены, за каждый домен твердит отдельная организация)
- надежность (дублирование серверов DNS)

Структура корневого домена: www(имя компьютера).wiki(домен второго уровня).com(домен верхнего уровня).(корневой домен)

Важная особенность - делегирование ответственности за хранение данных между доменами разных уровней.

Распределением имен занимаются регистраторы.

Режимы работы DNS:

1) Итеративный:

- если сервер отвечает за данную доменную зону - он возвращает ответ
- если нет - возвращает адрес DNS-сервера, у которого есть более точная информация

2) Рекурсивный:

- сервер сам выполняет запросы к другим серверам DNS, чтобы найти нужный адрес
Сервер разрешения имен DNS предоставляет провайдером/организацией, комп.
получает адрес локального DNS по DHCP.

Как только адрес найден, DNS сервер записывает его в кэш.

Типы ответа DNS:

- 1) авторитетный: ответ от сервера, обслуживающего доменную зону; получен из файлов на диске сервера
- 2) неавторитетный: ответ от сервера, не обслуживающего доменную зону; получен из кэша, данные могли устареть.

Работает по модели клиент-сервер. Взаимодействие идет в режиме запрос-ответ. DNS использует протокол UDP, номер порта 53.

Что еще может DNS:

- определять для доменного имени адреса IPv4 и IPv6
- задавать несколько доменных имен для одного IP-адреса
- находить адрес почтового сервера для домена
- определять IP-адрес и порт некоторых сетевых сервисов
- задавать адрес DNS-серверов для доменной зоны
- определять по IP-адресу доменное имя

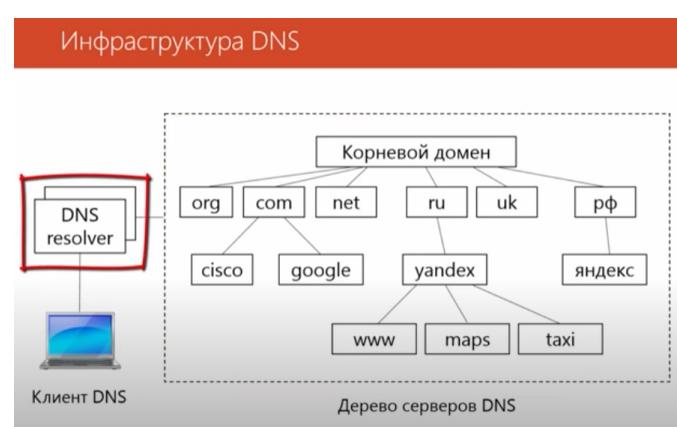
Запись MX нужна для отправки почты на почтовый сервер.

Чтобы присвоить несколько имен: псевдоним или несколько A записей.

Для некоторых сервисов можно задавать не только IP-адреса, но и номера портов. Для этого используется SRV запись.

Запись типа NS (name server) указывает адреса DNS серверов, отвечающих за зону.

Записи типа PTR используются реверсивной зоной для определения имени по IP.



Формат пакета DNS



- Протокол FTP (File Transfer Protocol) - протокол передачи файлов.

Работает по модели клиент-сервер. На сервере есть структура файлов. Клиент по протоколу подключается к серверу и работает с файлами.

Как и HTTP использует URL (`ftp://ftp-server.ru/path/file.format`). Использует два соединения: управляющее и для передачи данных.

Использует TCP, порт 21. Соединение данных: в активном режиме - порт сервера 20, в пассивном - порты больше 1024 (исп. если между клиентом и сервером есть интерфейс, напр. экран).

Есть аутентификация и режим анонимса.

Недостатки: проблемы с NAT (решает пассивный режим), низкая безопасность.

Замена: протоколы на основе SSH (SFTP, SCP).

Протокол FTP

Команда	Назначение
USER	Указать имя пользователя
PASS	Указать пароль
LIST	Просмотр содержимого каталога
CWD	Смена текущего каталога
RETR	Передать файл с сервера на клиент
STOR	Передать файл с клиента на сервер
TYPE	Установить режим передачи
DELE	Удалить файл
MKD	Создать каталог
RMD	Удалить каталог
PASV	Использовать пассивный режим
QUIT	Выход и разрыв соединения

6. Ур. представления (сообщение) - обеспечивает согласование синтаксиса и семантики передаваемых данных (форматы представления символов, форматы чисел). Шифрование и дешифрование. Пр.: Transport Layer Security (TLS), Secure Sockets Layer (SSL). Тут происходит кодирование и сжатие (пр.: JPEG, GIF).

Для защиты передаваемых по сети данных часто используется шифрование:

- Secure Socket Layer (SSL)
- Transport Layer Security (TLS)

Протоколы, которые используют SSL/TSL:

- HTTPS, порт 443
- IMAPS, порт 993
- SMTPS, порт 465
- FTPS

5. Сеансовый (сообщение) - позволяет устанавливать сеансы связи. Управляет диалогом (очередность передачи сообщений). Управляет маркерами (предотвращение одновременного выполнения критичной операции). Синхронизация (метки в сообщениях для возобновления передачи в случае сбоя).

Сеанс (сессия) - набор связанных между собой сетевых взаимодействий, направленных на решение одной задачи.

Загрузка Web-страницы:

- загрузка текста (.html)
- загрузка стилевого файла (.css)
- загрузка изображений

Подходы к загрузке Web-страницы:

- для каждого элемента создается отдельное соединение (HTTP 1.0)
- загрузка всех элементов через одно соединение TCP (HTTP keep-alive)

Пример взаимодействия на сеансовом уровне: аудио и видео конференция, т.к. на этом уровне устанавливается, каким кодаком будет кодироваться сигнал (кодаки должны совпадать с обеих сторон).

4. **Транспортный** (сегмент, дейтаграмма) - обеспечивает передачу данных между процессами на хостах. Надежность выше сети, гарант. порядок сообщений. TCP (все в точности) и UDP (может что-то потеряться). Сквозной уровень, т.к. дотавляет сообщения от источника адресату, в то время как предыдущие уровни исп. принцип звеньев (т.ж. тр. ур. называют сетенезависимым).

У хостов есть все 7 уровней, а вот у сетевого оборудования - первые 3. Транспортный уровень может связывать хосты, минуя сетевое оборудование - сквозное соединение. Важно указать адресацию, чтобы понимать, для какого процесса предназначен пакет. Для адресации используются порты. Это число от 0 до 65535. Каждое сетевое приложение на хосте имеет свой порт. Номера портов у приложений не повторяются. Формат записи: IP:порт.

Хорошо известные порты:

- 80 - HTTP (Web)
- 25 - SMTP (email)
- 53 - DNS
- 67, 68 - DHCP
- использовать может только root/ админ

Зарегистрированные порты: 1025-49151 (рег. в IANA). Динамические порты автоматически назначаются ОС клиенту сетевым приложениям.

Так, например, если мы (клиент) откроем браузер 1 и сделаем запрос на web-сервер (daemon), то сервер увидит наш IP и порт, и отправит ответ на порт браузера 1. Если откроем браузер 2 и сделаем запрос, ответ придет уже на порт браузера 2, который автоматически присвоила ему наша ОС при открытии.

Надежность уровня:

- гарантированная доставка данных:
 - подтверждение получения
 - данные отправляются снова, если не было подтверждения доставки
- гарантированный порядок следования сообщений - нумерация сообщений

Для взаимодействия с транспортным уровнем используется интерфейс сокетов:

Интерфейс транспортного уровня TCP/IP



UDP (User Datagram Protocol) - протокол действа грам пользователя. (дейтаграмма - сообщение UDP).

Особенности:

- нет соединения, за счет этого быстрее TCP
- нет гарантии доставки данных
- нет гарантии сохранения порядка сообщений

Зачем нужен? На транспортном уровне необходимо указать порты отправителя и получателя, что и делает протокол.

По надежности: в совр. сетях ошибки редки, да и обработать их могут приложения

Область применения: клиент-сервер и короткие запросы-ответы. Пример: DNS (клиент-DNS - сервер-DNS).

TCP (Transmission Control Protocol) - протокол управления передачей.

Обеспечивает надежную передачу потока байт (reliable byte stream).

Гарантии: доставка данных и сохранение порядка следования сообщений.

В протоколе TCP поток данных делится на отдельные сегменты. Они отправляются отдельно к получателю, который в свою очередь собирает их обратно в поток байт.

Чтобы обеспечить гарантию передачи данных TCP использует подтверждение получения.

Также TCP использует метод скользящего окна, подтверждая не каждый сегмент, а совокупность.

Для гарантии сохранения порядка отправления, TCP нумерует байты (сегментами по 1024

байт с 0 байт). Если произошла ошибка при отправке подтверждения о получении, получатель видит, что у него уже есть этот фрагмент и повторно отправляет подтверждение.

Перед передачей данных TCP необходимо установить соединение (это замедляет процесс).

Задачи соединения:

- убедиться, что отправитель и получатель хотят передавать друг другу данные
- договориться о нумерации потока байт
- договориться о параметрах соединения (макс. размер сегмента и т.д.)

После завершения передачи данных соединение разрывается.

Варианты подтверждения доставки:

- остановка и ожидание (WiFi, канальный ур.): отправка данных → ожидание подтверждения получения → подтверждение получения → продолжение отправки данных и т.д. (медленно, локальный)
- скользящее окно (TCP, транспортный уровень): отправка нескольких порций данных без ожидания подтверждения получения → кумулятивное подтверждение (получил последнюю порцию данных и все предыдущие) (быстро, для больших сетей)
Размер окна - кол-во байт данных, к-е могут быть переданы без получения подтверждения.

Еще есть выборочное подтверждение (Selective Acknowledgement, SACK) - подтверждение диапазонов принятых байт, эффективно при большом размере окна, доп. поле заголовка TCP (параметр).

Установление соединения:

- запрос на соединение (SYN) + порядковый номер передаваемого байта
- подтверждение соединения + информ. о предыдущих принятых байтах + ACK с номером ожидаемого байта
- подтверждение получения подтверждения о соединении с номером предыдущего полученного байта + ACK с номером следующего к передаче байта
- соединение установлено

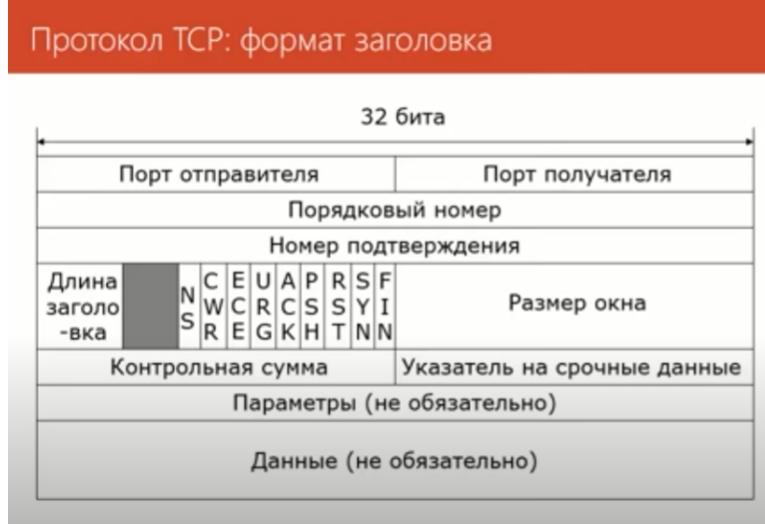
Разрыв:

- одновременное (обе стороны разорвали соединение)

- одностороннее (одна сторона прекращает передавать данные, но может принимать)

Есть варианты разрыва соединения: одностороннее (FIN) и из-за критической ситуации (RST, в обе стороны)

Порядок: FIN1 → ACK2 → (K2 закончил передачу данных) FIN2 → ACK1.



Управление потоком - справляется с проблемой “затопления”. Для этого есть поле “размер окна”.

Борьба с перегрузкой: окно перегрузки, которое рассчитывается в зависимости от нагрузки на сеть. В TCP есть AIMD (Additive increase/multiplicative decrease) - аддитивное увеличение/мультипликативное уменьшение - определяет размер окна перегрузки в зависимости от того, есть или нет перегрузки, опираясь на макс. размер сегмента. Сигнал о перегрузке - потеря сегмента или задержка сегмента или сигнал от маршрутизатора. Альтернатива - медленный старт: малый первоначальный размер окна перегрузки, при каждом подтверждении отправляется 1 сегмента, происходит экспоненциальный рост размера окна, а после сигнала о перегрузке все начинается с начала.

Интерфейс сокетов - для работы программиста с ПО на транспортном уровне (файлы).

Операции сокетов Беркли

Операция	Назначение
Socket	Создать новый сокет
Bind	Связать сокет с IP-адресом и портом
Listen	Объявить о желании принимать соединения
Accept	Принять запрос на установку соединения
Connect	Установить соединение
Send	Отправить данные по сети
Receive	Получить данные из сети
Close	Закрыть соединение

Работает по клиент-серверной модели. Сервер - программа, которая работает (слушает) на известном IP-адресе и порте и пассивно ждет запросов на соединение. Клиент - приложение, которое активно устанавливает соединение с сервером на заданом IP и порте.

NAT (Network Address Translation) - трансляция сетевых адресов (решение проблемы нехватки адресов IPv4). Это технология преобразования IP-адресов внутренней (частной) сети в IP-адреса внешней сети (Интернет).

Типы:

- статический (сколько комп=ов, столько и адресов, подх. для подключения к др. корп. сети)
- динамический (есть несколько IP для внеш. сети, которые поочередно используются комп-ами)
- один ко многим (masquerading) (один адрес на всех)

Межсетевой экран - отделяет сеть от других сетей (он же брандмауэр, firewall).

Перехватывает и проверяет пакеты.

Как обезопасить:

- флаг ACK - злоумышленник не сможет установить соединение (транспортный уровень)
- разрешить получение пакетов только от того, с кем установлено соединение (транспортный уровень)
- фильтрация на портах коммутатора по MAC-адресам (канальный уровень)

- прокси-сервер (прикладной уровень)
- фильтр содержимого (прикладной уровень)
- система обнаружения вторжений (IDS)
- система предотвращения вторжений (IPS)

Может замедлять и затруднять (а то и сделать невозможной) работу компьютера.

3. **Сетевой** (пакет)- объединяет сети, построенные на осн. разных технологий.

Обеспечивает: создание составной сети, согласование различий в сетях; адресацию (сетевые и глобальные адреса); опр. маршрута пересылки пакетов в сост. сети (маршрутизация). Оборудование: маршрутизатор. Использует IP адреса.

Объединяет сети разл. технологий (Ethernet, WiFi, 3/4/5G, MPLS). Этот уровень позволяет более удобно организовать масштабную сеть, чем с вифи и езернет.

Масштабируемость на сетевом ур.:

- агрегация адресов: работа с блоками адресов (блок адресов - сеть)
- запрет пересылки “мусорных” пакетов: т.е. тех, которые непонятно, куда отправлять
- возможность наличия неск. путей в сети: допускается неск. активных путей задача выбора лучшего пути - маршрутизация

Итого, сетевой уровень решает задачи: объединения сетей, маршрутизации, обеспечения качественного обслуживания.

Протоколы:

- **IP** - протокол передачи данных.

Передача данных: без гарантии доставки, без сохранения порядка следования сообщений. Протокол IP использует передачу данных без установки соединения.

Задачи: объединение сетей, маршрутизация, качество обслуживания.

Маршрутизация - поиск маршрута доставки пакета между сетями через транзитные узлы (маршрутизаторы).

Фрагментация - разделение пакета на несколько частей (фрагментов) для передачи по сети с маленьким MTU (max. transmission unit).

Формат заголовка IP-пакета

4 бита Номер версии	4 бита Длина заголовка	8 бит Тип сервиса	16 бит Общая длина			
		16 бит Идентификатор пакета	3 бита Флаги	13 бит Смещение фрагмента		
	8 бит Время жизни	8 бит Тип протокола	16 бит Контрольная сумма			
32 бита IP-адрес отправителя						
32 бита IP-адрес получателя						
Опции и выравнивание (не обязательно)						

- ICMP (управляющий протокол, доп.) - Internet Control Message Protocol - протокол межсетевых управляющих сообщений. Служит для управления сетью. Позволяет получать сообщения об ошибках сети (получатель недоступен, закончилось TTL - время жизни пакета, запрещено фрагментировать пакет). Служит для тестирования работы сети: ping (проверка доступности получателя) и traceroute (определение маршрута к получателю). Т.о. он нивелирует недостатки протокола IP (возможная потеря данных без дальнейшего восстановления передачи и без оповещения об ошибке). Есть утилита PING, к-я помогает обпределить наличие ошибок сети.

Типы ICMP- сообщений

Тип	Назначение сообщения
0	Эхо-ответ
3	Узел назначения недостижим
5	Перенаправления маршрута
8	Эхо-запрос
9	Сообщение о маршрутизаторе
10	Запрос сообщения о маршрутизаторе
11	Истечение времени жизни пакета
12	Проблемы с параметрами
13	Запрос отметки времени
14	Ответ отметки времени

Коды ICMP- сообщений (для типа 3)

Код	Причина
0	Сеть недостижима
1	Узел недостижим
2	Протокол недостижим
3	Порт недостижим
4	Ошибка фрагментации
5	Ошибка в маршруте источника
6	Сеть назначения неизвестна
7	Узел назначения неизвестен
8	Узел-источник изолирован
9	Административный запрет

- **ARP** (управляющий протокол, доп.) - Address Resolution Protocol - чтобы по IP определить протокол канального уровня. Протокол разрешения адресов. Связывает сетевой и канальный уровни. Позволяет определить по IP-адресу компьютера его MAC адрес.

Таблица соответствия (пр.: Linux “\$ cat /etc/ethers”): IP-MAC. В крупных сетях - ARP. Работает в режиме “широковещательный запрос-ответ компьютера, узнавшего свой IP”. Кеширование в ARP-таблицу (команда arp - a) на стороне отправителя.

Т.к. ниже сетевого уровня, пакеты ARP не проходят через маршрутизатор.

Зачем узнавать MAC-адрес? Сама технология (пр. Ethernet) не знает о том, что такое IP-адрес и для отправки информации ей нужен MAC.

- **DHCP** (управляющий протокол, доп.) - Dynamic Host Configuration Protocol - чтобы авто назначать IP комп-ам в составной сети. Протокол динамической конфигурации хостов. Модель “клиент-сервер”. Discover → Offer → Request → Ack (acknowledgement).

DHCP сервер должен быть установлен в той подсети, где находится клиент, т.к. пакеты DHCP не проходят через маршрутизатор (нет широковещания). Решение: DHCP-Relay сервер.

Устройство: маршрутизатор. Протоколы маршрутизации: BGP, OSPF, RIP, EIGRP.

2. **Канальный** (кадр, фрейм) - выделяет во входящем потоке бит отдельные сообщения, обнаруживает и корректирует ошибки. В широковещат. сетях еще обесп. физическую адресацию (указание, какому комп-у это нужно передать), а также управление доступом к среде передачи данных (в один момент вр. передает один комп.). Оборудование: коммутатор, точка доступа. Использует MAC адреса.

Передача данных: с сетевого уровня устройства 1 поступает пакет на канальный уровень устройства 1, канальный уровень присваивает пакету заголовок канального уровня и концевик, и через физический уровень передает этот кадр на канальный уровень устройства 2, который считывает заголовок и концевик, извлекает пакет сетевого уровня и передает своему сетевому уровню устройству 2.

Как канальный уровень определяет кадр (методы): указатель кол-ва бит (не на практике), вставка бит/байт (сейчас: протоколы HDLC и PPP: 1). 0(1x6)0 начало и конец кадра; 2). после 1ч5 в данных добавляется 0), устр-ва физ. уровня (преамбула Ethernet классического и избыточный код Fast Ethernet нового).

Подуровни: п/у управления логическими каналами (LLC) (передача данных, мультиплексирование, управление потоком, общий для разных технологий), п/у управления доступом к среде (MAC) (совместное использование разделяемой среды, адресация, специфичный для различных технологий, не я-ся обязательным).

Ethernet:

- на канальном и физическом ур.
- протокол STP (Spanning Tree Protocol): авто отключение дублир. соединений, связующее дерево, обесп. защиту от сбоев (широковещательного штурма при образовании кольца), надежность соед. между коммутаторами (новый - RSTP, rapid)

WiFi:

- на физ. и канал. ур, подуровни канал. ур.: п/у управл. логич. каналом (LLC) и п/у управления доступом к среде (MAC).
- типы кадров: кадры данных (передача данных), к. контроля (служебные и RTS, CTS, ACK), к. управления (реа-ия сервисов вифи, пр.: точка доступа)

1. **Физический** (бит)- передает биты единым потоком в виде сигналов по физ. каналу связи, не анализируя. Оборудование: концентратор. Здесь: Ethernet, WiFi, Bluetooth, инфракрасный порт. Сетевые устройства: концентраторы и репиторы.

Как данные переходят с канального на сетевой уровень?

- внутри одной сети:
 - есть K1 с IP1 и MAC1, K2 с IP2 и MAC2. В таблицу отправления вписываются данные отправителя, из данных получателя обычно известен только IP. По ARP узнаем его MAC. От получателя получаем ответ - такую же таблицу, только в зеркальном варианте.

- разные сети:
 - есть K1 с IP1 и MAC1, K2 с IP2 и MAC2. В таблицу отправления вписываются данные отправителя, из данных получателя обычно известен только IP. Проверяем, в одной ли сети: берем адрес сети отправителя и по ней проверяем по ней адрес получателя (используется маска подсети).
 - Т.к. в разных сетях, отправляем данные сначала на маршрутизатор, адрес которого знаем из таблицы маршрутизации. При помощи ARP узнаем MAC маршрутизатора. Маршрутизатор получает пакет и формирует свою таблицу для передачи: IP остаются, а MAC адреса меняются (отправитель - маршрутизатор с интерфейсом на стороне получателя, получатель - выявляем по ARP).
 - Получатель передает ответ - таблицу в которой IP-получатель - компьютер-отправитель, MAC-получатель - MAC сети, отправитель - данные отправителя. Получатель отправляет этот пакет на маршрутизатор.
 - Маршрутизатор снова меняет: отправитель имеет IP компьютера-получателя, а MAC - MAC сети со стороны отправителя; получатель - данные получателя пакета-подтверждения.

Итого: IP всегда остается, а MAC меняется.

Модель TCP/IP

Прим-ся на практик. (т.к. удобный стек протоколов)

4. Прикладной (сеансовый+представления+прикладной) - протоколы: HTTP, SMTP, DNS, FTP.
3. Транспортный - протоколы: TCP, UDP.
2. Интернет (сетевой) - протоколы: IP (доп.: ICMP, ARP, DHCP).
1. Сетевых интерфейсов (канальный+физический) - протоколы: Ethernet, WiFi, DSL.

Инкапсуляция - вкл-е сообщения вышестоящего ур. в сообщение нижестоящего. Т.е. по мере транспортировки данных с верхнего ур. до нижнего (от польз. интерфейса к машине). происх-т инкапсуляция, а обратно - декапсуляция (от машины к польз-ю). Сообщение = заголовок+данные+концевик.

IP адрес

Сетевой уровень.

Локальные адреса

- адреса в технологии канального уровня (пр.: MAC адрес в Ethernet, IMEI адрес в 4G)
- привязаны к конкретной технологии
- не могут быть использованы в гетерогенных сетях

Глобальные адреса

- адреса сетевого уровня (пр.: IP-адреса)
- не привязаны к технологии
- применяются при объединении сетей (Интернет)

Исп-ся для уникальной идентификации комп-ов в составной сети Интернет.

Версии протокола IP:

- IPv4: 4 байта
- IPv6: 16 байт

Сетевой уровень исп-т агрегацию адресов: масштабирование - работа не с отдельными адресами, а с подсетями. (поср-вом маршрутизаторов)

Подсеть - (IP-сеть, subnet) - множество комп-ов, у которых старшая часть IP-адреса одинаковая (октет - чать IP-адреса, отделенная точками).

Структура IP-адреса:

- номер подсети - старшие биты
- номер хоста (комп-а в сети) - младшие биты

Пример: 213.180.193.3 = 213.180.193 (номер подсети) + 0.0.0.3 (номер хоста)

Маска подсети - показывает, где в IP-адресе номер подсети, а где - хоста.

Структура маски:

- длина 32 бита
- единицы в позициях, задающих номер сети
- нули в позициях, задающих номер хоста

IP-адрес (дес.): 213.180.193.3

IP-адрес: 11010101.10110100.11000001.00000011

Маска: 11111111.11111111.11111111.00000000

Подсеть (через логическое И, т.е. AND): 11010101.10110100.11000001.00000000

Подсеть (дес.): 213.180.193.0

Маска подсети в виде префикса (ск-ко бит - номер подсети, остаток - номер хоста):
213.180.193.3/24

Типы IP-адресов: индивидуальный (unicast), групповой (multicast) и широковещательный (broadcast, есть ограниченное и направленное, на конце адреса - кол-во всех битов).

Работа эл. почты

В почтовом адресе используется доменное имя, для его определения почтовый сервис взаимодействует с DNS, используя MX записи. Чтобы посмотреть записи MX для домена можно использовать утилиту nslookup -type=mx gmail.com



Flask

Подробнее: <https://flask.palletsprojects.com/en/2.1.x/installation/#python-version>

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Tkinter GUI

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-tkinter>

Tkinter modules

Паттерны программирования

Команды:

- ls - посмотреть, в какой сейчас директории
- cd. /name - перейти в файл

Базы данных и Python

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-database-connectivity>

Git

Шпаргалка: <https://github.com/cyberspacedk/Git-commands>

Вопрос-ответ

- **Что такое NaN ?**

Nan - Not a Number - используется для представления записей, которые не определены, а т.ж. отсутствующих значений в наборе данных.

- **Что такое var?**

▼ **Как получить все аргументы функции (включая те, что не объявлены, но все-таки были переданы)?**

```
import inspect

def pair(boy, girl):
    print(boy+" and "+girl+" are a nice couple.")

pair(boy="John", girl="Lily")
a = inspect.getfullargspec(pair)
print(a)
```

→

John and Lily are a nice couple.

```
FullArgSpec(args=['boy', 'girl'], varargs=None, varkw=None, defaults=None, kwonlyargs=[], kwonlydefaults=None, annotations={})
```

- **Что такое чистая функция?**

Чистая функция — это функция, возвращаемое значение которой зависит только от передаваемых аргументов, без побочных эффектов. Проще говоря, если вы вызываете функцию n раз с n аргументами, и функция всегда возвращает одно и тоже значение, значит, она является чистой

- **Что такое деструктуризация?**

<https://codecamp.ru/blog/python-list-destructuring-aka-packing-and-unpacking/>

- **Что такое ООП? Из чего состоит?**

<https://smartiqa.ru/courses/python/lesson-6>

Объектно-ориентированное программирование — это подход, при котором вся программа рассматривается как набор взаимодействующих друг с другом объектов. При этом нам важно знать их характеристики

Инкапсуляция — скрытие поведения объекта внутри него. Объекту «водитель» не нужно знать, что происходит в объекте «машина», чтобы она ехала. Это ключевой принцип ООП.

Наследование. Есть объекты «человек» и «водитель». У них есть явно что-то общее. Наследование позволяет выделить это общее в один объект (в данном случае более общим будет человек), а водителя — определить как человека, но с дополнительными свойствами и/или поведением. Например, у водителя есть водительские права, а у человека их может не быть. (class extends)

Полиморфизм — это переопределение поведения. Можно снова рассмотреть «человека» и «водителя», но теперь добавить «пешехода». Человек умеет как-то передвигаться, но как именно, зависит от того, водитель он или пешеход. То есть у пешехода и водителя схожее поведение, но реализованное по-разному: один перемещается ногами, другой — на машине.

Переопределение каких-то методов у различных классов. К примеру изменения метода `toString()` для конкретного класса

- **Что такое абстракция?**

Абстракция - это способ создания простой модели, которая содержит только важные свойства с точки зрения контекста приложения, из более сложной модели. Иными словами - это способ скрыть детали реализации и показать пользователям только функциональность. Абстракция игнорирует нерелевантные детали и показывает только необходимые. Важно помнить, что мы не можем создать экземпляр абстрактного класса.

Всё программное обеспечение - это абстракция, скрывающая всю тяжелую работу и бездумные детали.

Многие программные процессы повторяются снова и снова. Поэтому, на этапе декомпозиции проблемы, мы удалим дублирование, записывая какой-либо компонент (функцию, модуль, класс и т. Д.), присваивая ему имя (идентификатор) и повторно используя его столько раз, сколько нам нужно.

Процесс декомпозиции - это процесс абстракции. Успешная абстракция подразумевает, что результатом является набор независимо полезных и перекомпонованных компонентов.

- **Какие еще парадигмы знаешь?**

1) Императивный стиль — это парадигма, основанная на составлении алгоритма действий (инструкций/команд), которые изменяют состояние (информацию/данные/память) программы. Фактически, программа на этих языках — это код, который выполняется компьютером сразу, без предварительной компиляции. Из языков высокого уровня, требующих компиляции исходного кода программы в машинный код (или интерпретации), к императивным можно отнести C, C++, Java.

2) Декларативный стиль — это парадигма, при которой описывается желаемый результат, без составления детального алгоритма его получения. В пример можно привести HTML и SQL. При создании HTML мы с помощью тегов описываем, какую хотим получить страничку в браузере, а не то, как нарисовать на экране заголовок статьи, оглавление и текст. В SQL, если нам нужно посчитать количество сотрудников с фамилией «Сидоров», мы напишем `SELECT count(*) FROM employee WHERE last_name = 'Сидоров';`. Тут ничего не сказано про то, в каком файле или области памяти находятся данные по сотрудникам, как именно выбрать из них всех Сидоровых и нужно ли вообще это делать для подсчёта их количества.

Парадигмы декларативного стиля:

Логическое программирование

В целом это скорее математика, чем программирование. Его суть заключается в том, чтобы, используя математические доказательства и законы логики, решать бизнес-задачи. Чтобы использовать логическое программирование, необходимо уметь переводить любую задачу на язык математики. Логическое программирование часто используется для моделирования процессов.

Функциональное программирование

Самая известная парадигма декларативного стиля — функциональное программирование. В этой парадигме понятие функции близко к математическому понятию функции. То есть это штука, которая как-то преобразует входные данные. Особенность функции в этой парадигме в том, что она должна быть чистой, то есть должна зависеть только от аргументов и не может иметь никаких побочных эффектов. Побочный эффект — это какое-либо изменение внешней среды. Если функция меняет глобальную переменную или, например, вызывает метод внешнего объекта, она меняет внешнюю среду. Это и есть побочный эффект.

- **Как проходит инициализация класса и экземпляра класса? Какие есть свойства (поля) класса? Какие есть методы экземпляра класса?**

<https://smartiqa.ru/courses/python/lesson-6> - про классы и объекты

```
class Character:  
    def __init__(self, mana, power):  
        self.m = mana  
        self.p = power  
  
warrior = Character(100, 200)  
print(warrior.m, warrior.p)
```

- **Что геттеры и сеттеры?**
- **Какие есть статичные методы?**
- **Наследование extends**
- **Родительский конструктор: super()**
- **Работа с файлами**
Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-file-io>
- **CSV в Python**
Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-csv>

- **Родительский класс: super**
- **Какие есть способы расширения функциональности класса?**
- **Что такое промисификация?**
- **Чем отличается HTTP2 от HTTP ?**

HTTP/2 (HTTP/2.0) – это бинарный протокол, в отличии от предыдущих версий. Это значит, что данные теперь занимают меньше места и быстрее обрабатываются.

Один из основных плюсов второй версии HTTP. В предыдущей версии для одного запроса была необходима установка отдельного TCP-соединения. И чем больше было запросов, тем медленней работал браузер. Благодаря мультиплексированию браузер отправляет сразу несколько запросов через одно TCP-соединение.

Приоритизация

Первый вид приоритизации подразумевает получение потоком определенного веса, который дальше распределялся между потоками, чтобы нагрузка была равномерной. Данный тип выбора приоритета впервые был применен в протоколе SPDY.

Второй вариант является основным для http/2, а его суть заключается в том, что браузер запрашивает у сервера отдельную загрузку некоторых элементов контента, к примеру, сначала скриптов JavaScript, а затем изображений.

Если сравнивать его с прошлой версией протокола, то здесь разработчики поменяли методы распределения данных на фрагменты и их отправку от сервера к пользователю и наоборот. Новая версия протокола позволяет серверам доставлять информацию, которую клиент пока что не запросил. Это было внедрено с той целью, чтобы сервер сразу же отправлял браузеру для отображения документов дополнительные файлы и избавлял его от необходимости анализировать страницу и самостоятельно запрашивать недостающие файлы.

Еще одно отличие http 2.0 от версии 1.1 – мультиплексирование запросов и ответов для решения проблемы блокировки начала строки, присущей HTTP 1.1. Еще в новом протоколе можно сжимать HTTP заголовки и вводить приоритеты для запросов.

- **Что такое CSRF, XSS?**

CSRF (англ. cross-site request forgery — «межсайтовая подделка запроса», также известна как XSRF) — вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Если жертва заходит на сайт, созданный злоумышленником, от её лица тайно отправляется запрос на другой сервер (например, на сервер платёжной

системы), осуществляющий некую вредоносную операцию (например, перевод денег на счёт злоумышленника). Для осуществления данной атаки жертва должна быть аутентифицирована на том сервере, на который отправляется запрос, и этот запрос не должен требовать какого-либо подтверждения со стороны пользователя, которое не может быть проигнорировано или подделано атакующим скриптом.

XSS (англ. Cross-Site Scripting — «межсайтовый скрипting») — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «Внедрение кода».

- **Что такое DOM?**

DOM – это представление HTML-документа в виде дерева тегов.

▼ Пример

```
<!DOCTYPE HTML>
<html>
<head>
<title>О лосях</title>
</head>
<body>
Правда о лосях.
</body>
</html>
```

- **Что такое webApi ?**

- 1). Какое определение можно дать для WEB API и зачем он нужен?

Веб-API - это интерфейс прикладного программирования для веб-сервера или веб-браузера. Это концепция веб-разработки, обычно ограниченная клиентской стороной веб-приложения (включая любые используемые веб-фреймворки), и поэтому обычно не включает детали реализации веб-сервера или браузера, такие как SAPI или API, если они не доступны для общего доступа через удаленное веб-приложение.

Серверный веб-API - это программный интерфейс, состоящий из одной или нескольких общедоступных конечных точек для определенной системы сообщений запрос-ответ, обычно выраженной в JSON или XML, которая предоставляется через Интернет - чаще всего посредством HTTP веб сервера.

Гибридные приложения - это веб-приложения, сочетающие в себе использование нескольких серверных веб-API.

Веб-хуки - это серверные веб-API, которые принимают входные данные в виде универсального идентификатора ресурса (URI), который предназначен для использования в качестве удаленного именованного канала или типа обратного вызова, так что сервер действует как клиент для разыменования предоставленного URI и запуска событие на другом сервере, который обрабатывает это событие, тем самым обеспечивая тип однорангового IPC.

2). Можно ли сказать что если сервер на POST или GET запрос возвращает в ответ контент в формате JSON, то это у меня WEB API?

Да

3). Являются ли web-сервисами, например WCF, WEP API?

WCF и ASP.NET Web API - это фреймворк/библиотека с помощью которой вы можете организовать работу WEB-API в вашем приложении.

- **Что такое рекурсия и чем она опасна?**
- **Что будет при обработке тяжелой вычислительной функции (например цикла суммирования) и как решить возникающие проблемы?**
- **Что такое генераторы и итераторы? Что такое yield?**
- Какие есть режимы работы с файлами?

a - добавить что-то в конец файла и создать файл, если его еще нет

w - очищает файл и записывает информацию заново

r - просто чтение (если файла нет - ошибка)

- **Как устроен сборщик мусора в Python?**
Как только объекты больше не нужны, Python автоматически освобождает память из под них. Подробнее: <https://habr.com/ru/post/417215/>.
- **Возможные виды утечек памяти и как с ними бороться?**
- **Про прокси**
- **Что такое SOLID? KISS, DRY, YAGNI?**

SOLID:

- **Single Responsibility Principle** - для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.
- **Open-Closed Principle** - программные сущности ... должны быть открыты для расширения, но закрыты для модификации.
- **Liskov Substitution Principle** - объекты в программе должны быть заменяемыми на экземпляры их подтипов без изменения правильности выполнения программы.
- **Interface Segregation Principle** - много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения.
- **Dependency Inversion Principle** - зависимость на Абстракциях. Нет зависимости на что-то конкретное.

KISS (Keep It Simple Stupid) - утверждает, что большинство систем работают лучше всего, если они остаются простыми, а не усложняются.

DRY (Don't Repeat Yourself) - это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода, особенно в системах со множеством слоёв абстрагирования.

YAGNI (You Aren't Gonna Need It) - процесс и принцип проектирования ПО, при котором в качестве основной цели и/или ценности декларируется отказ от избыточной функциональности, — то есть отказ добавления функциональности, в которой нет непосредственной надобности.

- **Что такое унарный, бинарный, тернарный оператор?**
- **Что такое вычислительная сложность?**
- **Что такое область видимости?**

Обычно, по области видимости, переменные делят на глобальные и локальные. Глобальные существует в течении всего времени выполнения программы, а локальные создаются внутри методов, функций и прочих блоках кода, при этом, после выхода из такого блока переменная удаляется из памяти.

В Python их 4:

- 1). Local - эту область видимости имеют переменные, которые создаются и используются внутри функций.
- 2). Enclosing - суть данной области видимости в том, что внутри функции могут быть

вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в enclosing области видимости.

- 3). Global - Переменные области видимости global – это глобальные переменные уровня модуля (модуль – это файл с расширением .py).
- 4). Built-in - уровень Python интерпретатора. В рамках этой области видимости находятся функции open, len и т.п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. Built-in – это максимально широкая область видимости.

- **Что такое lambda функции (стрелочные функции)?**
- **Что такое мемоизация и каррирование в Python?**

▼ Мемоизация – способ оптимизации при котором сохраняется результат выполнения функции и этот результат используется при следующем вызове. (дело тут, по сути, в быстроте выполнения).

```
from functools import lru_cache #LRU - Least Recently Used
```

```
@clock  
@lru_cache()  
  
def fib(n):  
    if n < 2:  
        return n  
  
    return fib(n-2) + fib(n-1)  
  
print('fib(20) =', fib(20))
```

▼ Каррирование – это преобразование функции от многих аргументов в набор функций, каждая из которых является функцией от одного аргумента. Мы можем передать часть аргументов в функцию и получить обратно функцию, ожидающую остальные аргументы.

```
def greet_deepliy_curried(greeting):  
  
    def w_separator(separator):  
        def w_emphasis(emphasis):  
            def w_name(name):  
                print(greeting + separator + name + emphasis)  
  
            return w_name  
  
        return w_emphasis  
  
    return w_separator
```

```
    return w_name

    return w_emphasis

return w_separator

greet = greet_deepliy_curried("Hello")("...")(".")

greet('German')
greet('Ivan')
```

————через lambda————

```
greet_deepliy_curried =lambda greeting: lambda separator: lambda emphasis:
    lambda name: \
        print(greeting + separator + name + emphasis)
```

- **Что такое менеджер контекста в Python?**
- **Что такое частичное применение функции в Python?**

▼ Это процесс применения функции к части ее аргументов. Другими словами, функция, которая принимает функцию с несколькими параметрами и возвращает функцию с меньшим количеством параметров. Подробнее:
<https://habr.com/ru/post/335866/>.

```
from functools import partial

def greet(greeting, separator, emphasis, name):
    print(greeting + separator + name + emphasis)

newfunc = partial(greet, greeting='Hello', separator=',', emphasis='.')
newfunc(name='German')
newfunc(name='Ivan')
```

- **Что такое замыкание в Python?**

Замыкание (*closure*) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами. Подробнее:
<https://devpractice.ru/closures-in-python/>.

▼ Пример

```
>>> def fun1(a):
    x = a * 3
    def fun2(b):
        nonlocal x
        return b + x
    return fun2

>>> test_fun = fun1(4)

>>> test_fun(7)
19
```

- **Что такое поверхностное и глубокое копирование в Python?**

▼ Поверхностное (**shallow**) копирование - также создает отдельный новый объект или список, но вместо копирования дочерних элементов в новый объект, оно просто копирует ссылки на их адреса памяти. Следовательно, если вы сделаете изменение в исходном объекте, оно будет отражено в скопированном объекте, и наоборот.

```
# Program 2 - Shallow Copy

import copy

result_A = [95, 85, 82]

result_B = copy.copy(result_A)

print(result_A)

print(result_B)
```

▼ Глубокое (**deep**) копирование - создает новую и отдельную (независимую) копию всего объекта или списка со своим уникальным адресом памяти.

```
# Program 1 - Deep Copy

import copy

result_A = [90, 85, 82] # Student A grades

result_B = copy.deepcopy(result_A) # Student B grades (copied from A)

print(result_A)

print(result_B)
```

- **Что такое магические методы класса в Python?**

- **Что такое деструктуризация?**

Деструктуризация - распаковка, т.е. разбиение целого итерабельного объекта(например, списка) на части. Подробнее: <https://codecamp.ru/blog/python-list-destructuring-aka-packing-and-unpacking/>.

- **Что такое функциональное программирование в Python?**

- **Что такое магические методы в Python?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-magic-methods>

- **Какие есть ключевые слова в Python?**

- **Что такое регулярные выражения?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-regex>

- **Как реализуется асинхронность в Python?**

- **Как Python взаимодействует с REST API?**

- **Что такое модуль mimetypes в Python?**

- **Как происходит обработка событий в Python?**

- **Что такое worker в Python?**

- **Что такое стек вызовов в Python?**

- **Хранение данных в backend**

- **Как работают токены в Python?**

- **Что такое унарный, бинарный, тернарный операторы Python?**

- **Что такая вычислительная сложность в Python?**

- **Как работает обработка ошибок и исключений в Python?**

- **Что такое polyfill в Python?**

- **Что такое socket module?**

Подробнее: <https://www.knowledgehut.com/tutorials/python-tutorial/python-socket-module>

▼ Вопросы на собеседовании

1. В чем разница между модулем и пакетом в Python?

- 2. Какие встроенные типы доступны в Python?**
- 3. Что такое лямбда-функция в Python?**
- 4. Что означает пространство имен?**
- 5. Объясните разницу между списком и кортежем?**
- 6. Чем отличается pickling от unpickling?**
- 7. Что такое декораторы в Python?**
- 8. Разница между генераторами и итераторами?**
- 9. Как преобразовать число в строку?**
- 10. Как используется оператор // в Python?**
- 11. Есть ли в Python инструкция Switch или Case, как в C?**
- 12. Что такое функция range() и каковы ее параметры?**
- 13. Как используется %s?**
- 14. Обязательно ли функция Python должна возвращать значение?**
- 15. Есть ли в Python функция main()?**
- 16. Что такое GIL?**
- 17. Какой метод использовался до оператора «in» для проверки наличия ключа в словаре?**
- 18. Как изменить тип данных списка?**
- 19. Каковы ключевые особенности Python?**
- 20. Объясните управление памятью в Python**
- 21. Что такое PYTHONPATH?**
- 22. Чувствителен ли Python к регистру?**
- 23. Объясните использование функций `help()` и `dir()`.**
- 24. Что такое модули Python?**
- 25. Объясните, что означает «self» в Python.**
- 26. Что такое инженерия и процесс разработки в целом?**

27. Какие знаете принципы программирования?
28. Чем отличаются процедурная и объектов-ориентированная парадигмы программирования?
29. Какие основные принципы ООП (наследование, инкапсуляция, полиморфизм)?
30. Что такое множественное наследование?
31. Какие есть шесть этапов разработки продукта в Software Development lifecycle и какая разница между Agile и Kanban?
32. Какие есть методы HTTP-запросов и какая между ними разница?
33. Как выглядят HTTP-request / response?
34. Что такое авторизация и как она работает?
35. Что такое cookies?
36. Что такое веб уязвимость?
37. Какие знаете классические базы данных?
38. Как читать спецификацию в конкретном языке (например, PEP8 в Python)?
39. Как происходит взаимодействие клиента и сервера?
40. Какие есть подходы к проектированию API?
41. Как использовать паттерны программирования?
42. Что такое Acceptance Testing и зачем его используют?
43. Что такое модульные и интеграционные тесты, API-тесты?
44. Как писать unit-тесты?
45. Какие есть best practices в написании автотестов?
46. Какие базовые команды системы контроля версий?
47. Как использовать Git?
48. В чем разница между хешированием и шифрованием?
49. Python - интерпретируемый язык или компилируемый?
50. Какие есть меняющиеся и постоянные типы данных?

51. Что такое область видимости переменных?
52. Что такое introspection?
53. Разница между `is` и `==`?
54. Разница между `__init__()` и `__new__()`?
55. В чем разница между потоками и процессами?
56. Какие есть виды импорта?
57. Что такое класс, итератор, генератор?
58. Что такое метакласс, переменная цикла?
59. В чем разница между итераторами и генераторами?
60. В чем разница между `staticmethod` и `classmethod`?
61. Как работают декораторы, контекстные менеджеры?
62. Как работают `dict comprehension`, `list comprehension` и `set comprehension`?
63. Можно ли использовать несколько декораторов для одной функции?
64. Можно ли создать декоратор из класса?
65. Какие есть основные популярные пакеты (`requests`, `pytest`, etc)?
66. Что такое lambda-функции?
67. Что означает `*args`, `**kwargs` и как они используются?
68. Что такое `exceptions`, `<try-except>`?
69. Что такое PEP (Python Enhancement Proposal), какие из них знаете (PEP 8, PEP 484)?
70. Напишите hello-world сервис, используя один из фреймворков.
71. Какие есть типы данных и какая разница между `list` и `tuple`, зачем они?
72. Как использовать встроенные коллекции (`list`, `set`, `dictionary`)?
73. В чем заключается сложность доступа к элементам `dict`?
74. Как создается объект в Python, для чего `new`, зачем `init`?
75. Что знаете из модуля `collections`, какими еще `built-in` модулями пользовались?

76. Что такое шаблонизатор и как в нем выполнять базовые операции (объединять участки шаблона, выводить дату, выводить данные с серверной стороны)?
77. Как Python работает с HTTP-сервером?
78. Что происходит, когда создается виртуальная среда?
79. Какие есть базовые методы работы с SQL-базой данных в Python?
80. Что такое SQL-транзакция?
81. Как сделать выборку из SQL-базы с простой агрегацией?
82. Как выглядит запрос, который выполняет JOIN между таблицами и к самим себе?
83. Как отправлять запросы в SQL-базу данных без ORM?
84. Что такое алгоритмы (например, Big-O notation)?
85. Какие есть базовые алгоритмы сортировки?
86. Что такое Bubble Sort и как это работает?
87. Что такая линейная сложность сортировки?
88. Ориентируетесь ли в *nix, можете ли написать скрипты/автоматизацию для себя и коллег?
89. Что такое многопоточность?
90. Что такое архитектура веб сервисов?
91. Как работает современное нагруженное веб приложение (нарисовать и обсудить примерную архитектуру, например, Twitter или Instagram)?
92. Что нужно для сайта / сервиса среднего размера (redis \ celery \ кэш \ логирование \ метрики)?
93. Как написать, задеплоить и поддерживать (микро) сервис?
94. Как масштабировать API?
95. Як проводить Code review?
96. Что такое абстрактная фабрика, как ее реализовать и зачем ее применяют?
97. Что такая цикломатическая сложность?
98. Async Python: как работает, зачем, что под капотом?

99. Сравнить асинхронные web-фреймворки.
100. Что такое модель памяти Python?
101. Что такое SQLAlchemy (Core и ORM частей) и какие есть альтернативы?
102. Принципы работы и механизм Garbage collection, reference counting?
103. Как работает thread locals?
104. Что такое *slots*?
105. Как передаются аргументы функций в Python (by value or reference)?
106. Что такое type annotation?
107. Для чего используют нижние подчеркивания в именах классов?
108. Статические анализаторы: Flake8, Pylint, Radon.
109. Разница между SQL и NoSQL?
110. Как оптимизировать SQL-запросы?
111. Какие есть уровни изоляции транзакций?
112. Какие есть виды индексов?
113. Точечные вопросы по выбору БД, движков БД?
114. **Front-end:** есть ли опыт работы с «современным» JS (Babel, Webpack, TS, ES)?
115. **DevOps:** работали ли с Docker-контейнерами, объяснить основные термины K8s (кластер, pod, node, deployment, service), что такое Kibana?
116. **Алгоритмы:** что такое времененная сложность алгоритма (time complexity)?
117. **Углубленные знания Linux:** как зайти на внешний сервер, работать с пакетами, настроить среду и выполнять операции?
118. **Специфично для Data Science:** как работать с пакетами для обработки и визуализации данных (NumPy, Pandas и другие)?
119. Что такое @property?
120. Каким образом можно запустить код на Python параллельно?
121. Как работать с stdlib?

122. Какие задачи решали с помощью метаклассов?
123. Что такое дескрипторы?
124. Знания других языков, кроме Python (опыт).
125. Какие технологические особенности реализации распределенных систем?
126. Какие есть низкоуровневые особенности языков и фреймворков?
127. Способы и методы управления памятью.

Загуглить:

- MongoDB
- Redis
- Tarantool
- Kafka
- декомпозиция
- веб-сервисы
- сетевые протоколы
- ui-фреймворки
-

1. Основы (как работает интернет):

Как работает Интернет?

Центр обработки данных (например, гугловый) м. б. в тысячах км от хоста – на нем хранятся данные.

Сеть волоконно-оптических кабелей охватывает весь мир и соединяют центр обработки данных и конкретный хост.

Мобильная сеть или маршрутизатор вай-фай – посредники между хостом и оптоволоконной сетью.

Конкретный файл, который необходимо передать хранится в центре обработки данных, а точнее на твердотельном накопителе (SSD), который выполняет функции внутренней памяти сервера.

Каждое устройство, подключенное к сети, имеет IP-адрес (его присваивает устройству интернет-провайдер). Кстати, сервер в ЦОД также имеет IP-адрес (если его узнать, сайты, которые на нем хранятся станут доступны).

IP-адресу соответствует доменное имя для удобства запоминания пользователем.

Сервер хранит несколько сайтов одновременно. однако они не могут быть привязаны к одному IP-адресу. Для разрешения этого неудобства используются заголовки хоста.

Чтобы серверу легче было соотнести IP-адрес с его доменным именем существует DNS-сервер.

DNS-сервером могут управлять интернет провайдеры и др. компании.

Сайт состоит из файлов, которые хранятся на сервере.

Как проходит поиск сайта:

1. Вводим в адресной строке домен
2. Браузер отправляет запрос на DNS-сервер
3. DNS-сервер ищет соответствующий IP-адрес и отдает его браузеру
4. Браузер перенаправляет запрос с IP-адресом в ЦОД (т.е. к серверу, на котором хранится сайт)
5. Сервер получает запрос на доступ к сайту, и начинается поток данных. Они передаются в цифровом формате (последовательности 0 и 1, которые разделены на пакеты, каждый по 6 битов, в каждом из которых также есть порядковый номер и IP-адрес хоста) через оптоволоконный кабель в виде световых импульсов.
6. Маршрутизатор (модем, например) преобразует световые сигналы оптоволоконного кабеля в электрические. А для передачи эл. сигналов на хост. ноутбук, в частности, используется кабель Ethernet, а с мобильной связью сигналы из оптоволокна направляются на вышку сотовой связи, с которой сигнал в виде электромагнитных волн передается на смартфон.
7. Браузер (он же клиент) получает данные и отрисовывает веб-страницу

Организация ICANN занимается регистрацией доменных имен, контролем IP-адресов и т.д.

Для скорой передачи данных каждый пакет выбирает кратчайший доступный маршрут, а там собираются в соответствии с порядковыми номерами, если часть информации не достигла хоста, с него отправляется повторный запрос на отправку данных.

Для управления потоками данных существуют протоколы (http/https – веб доступ, TCP/IP – передача данных, RTP – медиа, т.е. онлайн-видео, онлайн стримы, IP-телефония). Для разных приложений протоколы различны (исходя из нужд).

Frontend - клиентская часть сайта (та, которую мы видим, браузер, для ее создания используются CSS – каскад стилей для HTML, HTML - разметка, JavaScript). Backend – часть, связанная с сервером БД (общение происходит посредством серверных языков программирования – JavaScript, Python, Ruby, PHP, SQL)

MAC-адрес (media access control) – правила именования устройств на канальном уровне (для устройств: сетевых плат и т.д.). Называет их компания-изготовитель.

К IP-адресу добавили битовую маску. Есть биты, логические операции (пример & - если и то и то True, т.е. 1). IP-адрес состоит из 4 байт (IPv4), каждый из 8 бит. После применения операции – другое число, чтобы определить, какие биты исп. для нумерации сети и узлов внутри сети. Пр.: число до – к сети, после – к узлу.

У узла есть ИП и фикс. маска подсети. Для него внутренние адреса – номер сети ИП совпадает – узлы локальной сети, непосредственно доступны. Внешние узлы – через шлюз/маршрутизатор (у него один адрес в одной сети, а другой – в другой).

Таблица маршрутизации: номер сети назначения (IP через маску), IP-адрес шлюза, метрика.

Как работает ОС

Процессор:

- Читает инфу из памяти
- Выполняет операцию по инструкции
- Кладет в память результат

BIOS – basic input-output system - устройство ввода/вывода - первый интерфейс. Т.е. у компа теперь 2 режима: устройство вычислений и устройство ввода/вывода (делает удобней).

ОС отделяет приложения друг от друга и от себя, чб при поломке одного остальные не страдали.

Позволяет получить абстракции, облегчающие работу.

Абстракции:

- процессы и потоки
- файлы и файловые системы
- адресное пространство и память
- сокеты, протоколы, устройства и т.д.

Интерфейс системных вызовов

Kernal Space (OS) (– вызовы выполняются внутри ядра) и User Space (- а вызываются снаружи) – разграничивает процессор.

BASH – интерпретатор командной строки.

Ядро:

- обрабатывает запросы приложений
- обрабатывает запросы оборудования
- обеспечивает диспетчеризацию процессов (scheduling)
- обрабатывает исключительные ситуации

Сервисы, предоставляемые ОС:

- управление процессами
- управление памятью
- файлы, их содержимое, каталоги и директории
- модель безопасности
- межпроцессное взаимодействие
- прочее: терминалы, сети, таймеры, периферия

Драйвер – превращение интерфейса программы в общий интерфейс

Процесс – абстракция, которая предоставляет иллюзию ПК (как будто все ресурсы направлены на пользователя или процесс)

- адресное пространство (страницы, таблицы)
- CPU
- файлы и др. абстракции ОС
- состояние (состояние в памяти и набор регистров внутри процессора)

- стек

Поток – поток исполнения инструкций, процесса или его части

TGID – про поток

PID – про процесс

Вытесняющая многозадачность

Прерывание – ситуация остановки процессором последовательного выполнения программы для выполнения запроса или реакции на событие.

Системный вызов – специальное программное прерывание, соответствующее запросу сервера у ядра.

Иключение – неверное действие программы, приводящее к генерации прерывания.

Системные вызовы и драйверы

System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n bytes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

Многозадачность:

- прерывание таймера
- смена контекста
- план блокировок (при наличии нескольких CPU)
- освобождение ресурсов при завершении процесса

состояние процессора хранится в регистрах

машинные коды – атомарные, их не видно среднестатистическому прогеру

deadlock – когда есть 2 критических процесса, которые происходят в критической секции и задействуют критические ресурсы

Active – состояние, когда процесс выполняется на процессоре

Waiting – состояние, когда процесс вызывает операцию ввода-вывода, которое не может сразу выполнено

Ready - когда операция завершилась, готова к исполнению, но не выполнится

Физическая память

Виртуальная память (пейджинг, страничная организация памяти)

Процессор:

x86 – архитектура в большинстве компьютеров (32 и 64 бита)

ARM - архитектура в мобильных устройствах

AVR – авто, телевизоры и т.д.

Для каждой архитектуры пишется ПО.

Разрядность – величина, которая определяет размерность машинного слова (макс. кол-во бит, к-ми может оперировать процессор за раз)

Команду процессор считывает из памяти и кладет куда-то. Регистровая память (временные результаты вычислений, быстрая) и кэш память. Максимальный размер регистра = разрядности.

Регистр д.б. кратен ячейке ОЗУ. Байтовая адресация – каждая ячейка = 1 байту и каждый байт имеет свой адрес, по которому процессор может к нему обратиться. Словесная адресация – то же самое, но размер ячейки = машинному слову, и процессор обр-ся к машинному слову.

ASCII-символ = 7 бит

Регистры специального назначения – предназначаются для конкретного содержимого. В сегментных регистрах хранятся адреса памяти; регистры для работы со стеком указывают на каналы фрейма и верхушку стека; флаговые регистры содержат различные биты, которые отражают состояние результата предыдущей операции; указатель команд (instruction pointer) указывает адрес команды, которую нужно выполнить следующей и др.

Регистры общего назначения – могут быть использованы на усмотрение прогера, однако есть опр. соглашения по работе компиляторов. Выступают в роли переменных, параметров, временного хранилища для результатов вычислений.

Язык ассемблера – низкоуровневый язык программирования, состоящий из символического обозначения машинных команд.

Ассемблер – программа-транслятор этого языка в машинный код.

При включении компьютер запускает биос, его задачи: найти первое дисковое устройство, которое было указано, взять оттуда первый сектор, загрузить его в память и передать на него управление (указать процессору на то, чтобы теперь он начал выполнять команды по адресу этой загруженной программы, а именно - ОС). ОС переводит процессор в безопасный режим. ОС выставляет специальный флаг системного регистра, устанавливает разрешения и условия для других программ, распределяет таблицы дескрипторов и прерываний. Т.о. ОС становится полноценным хозяином компа. Далее реализуется работа с памятью, выставляются различные ограничения и т.д.

Когда мы запускаем программу, ОС выделяет под нее место, загружает ее в ОЗУ (например, жесткого диска), после чего передает ей управление, т.е. говорит процессору, с какого места из памяти ему нужно выполнить следующую команду. На этом моменте используется регистр IP. После процессор обновляет значение в этом регистре, чб он указывал на следующую инструкцию в памяти. Т.е. он определяет размер инструкции и прибавляет его к значению в IP.

Режимы работы процессора:

- реальных адресов – 16-битный режим, в который процессор переходит сразу после включения компьютера (адрес, сформированный программами, я-ся реальными не требует преобразований)
- защищенный – 32-битный режим, в который можно перейти только из режима реальных адресов, при нем обеспечивается защита данных, ОС, прикладных программ и данных этих программ друг от друга благодаря разделению приложений на разные уровни привилегий
- 64 разрядный режим – в него можно перейти только из защищенного режима

В оф. док-ии:

- режим реальных адресов + защищенный – legacy mode
- 64 разрядный + режим совместимости (поддержка 32 и 16-разрядного кода) – long mode

Уровни привилегий – доступ к использованию ресурсов процессора (начинаются с защищенного режима).

0 уровень – полный доступ к процессору (на нем ОС)

- 1 уровень – на нем дей-т запреты с 0 уровня
- 2 уровень – запреты с 0 и 1 ур.
- 3 уровень – (пользовательский, на нем прикладные проги) запреты с 0, 1 и 2 ур.

Устройство адресации. Вся оперативная память поделена на сегменты. Их размер зависит от режима, в котором работает процессор. Ячейки представляются в специальном формате: логический адрес = адрес начала сегмента (16 бит) : смещение в сегменте (16 бит).

В режиме реальных адресов адрес делится на сегменты по 64 кб.

Логический адрес преобразуется в 20 битный адрес ячейки памяти: физический адрес = адрес начала сегмента << 4 + смещение сегмента.

Максимальный размер адресов – 1мб физической памяти.

В защищенном режиме память делится на сегменты от 0 до 4 гб. ОС создает иллюзию того, что каждой программе доступно все пространство памяти. Она активирует механизм трансляции виртуальных адресов в физические, т.е. программы начинают работать в виртуальном адресном пространстве, про которое она сами не догадывается. Они продолжают формировать логические адреса, как и раньше, предполагая, что обращаются к реальной памяти.

Логический адрес, созданный программой в этом режиме, преобразуется не в физический, а в виртуальный: логический адрес = селектор дескриптора (16 бит) + смещение в сегменте (32 бита).

Селектор дескриптора – индекс дескриптора: уровень привилегий + таблица + индекс. Таблица и индекс показывают GDT/LDT (начальный адрес сегмента, уровень привилегий, размер сегмента).

Виртуальный адрес (32 бита) = адрес начала сегмента + смещение в сегменте.

Страницчная организация памяти – структура, представляющая собой виртуальную память. Все адресное пространство разбивается на страницы, чей размер зависит от режима работы процессора и режима трансляции адресов. Каждая страница описывается структурой, которая объединяется в таблицу страниц, а таблицы страниц объединяются в каталог страниц. Виртуальный адрес = индекс в каталоге страниц + индекс в таблице страниц + смещение в странице.

При помощи механизма трансляции адресов виртуальный адрес преобразуется в физический (32 бита), максимальное число – 4 гб физической памяти.

В 64-битном режиме в основном так же, но есть различия, например, практически полностью отключена сегментация. В физической памяти доступно до 2^{52} байт, в виртуальной – до 2^{48} в виртуальной.

Ограничения обусловлены архитектурой процессора и ОС.

Как внешние устройства взаимодействуют с процессором, если он всегда занят своей работой?

Например, нам нужно, чтобы процессор обработал запрос пользователя с клавиатуры прямо во время работы программы и без задержек. Внешнее устройство (клавиатура) через контроллер прерываний передает процессору сигнал о прерывании, чтобы он прервал выполнение текущей программы и передал управление специальной функции – обработчику прерываний. Конечно, сначала сохраняется состояние текущей программы (регистры, адрес сл. инструкции и т.д.). После совершения работы обработчик передает управление программе, восстанавливая ее последнее сохраненное состояние, и она продолжает работу с того места, на котором была прервана.

Прерывания находятся в специальной таблице дескрипторов прерываний, у которых есть уровни привилегий, которые определяет уровень привилегий программы, которая определяет прерывание.

Виды прерываний:

- программные – генерирует сам программист-оператор в ряде ОС (в винде нельзя, есть спец. проги)
- аппаратные – генерируются контроллером прерываний, которые выполняются согласно приоритетам
- исключения – их генерирует сам процессор при попытке программы нарушить ограничения защиты или при возникновении ошибок в ходе ее выполнения

Механизм многозадачности основан на прерывании – одновременная работа многих программ достигается путем попеременного их переключения.

При реализации ОС программы разделяются на процессы, которые разделяются на потоки/задачи (задача – самостоятельная последовательность команд, которая выполняется в своем окружении).

Многопроцессорность. Каждый процессор оснащен несколькими ядрами, а серверы оснащены несколькими процессорами.

Типы многопроцессорных систем:

- на материнской плате несколько сокетов для процессоров
- когда процессор имеет несколько ядер
- когда процессор имеет виртуальные ядра

Frontend

клиентская часть

Пользовательский интерфейс для общения с сервером. UI/UX

Пул компетенций: HTML, CSS, JS, Git, WebPack, Gulp, SASS, LESS, JQuery, React, View, базовые навыки PS, Figma, Avacode и т.д.

Верстальщик – макеты дизайна, HTML, CSS (логика, JS и прочее).

Backend

программно-аппаратная часть

Хостинг

Это услуга, предоставляемая компаниями, заключающаяся в предоставлении определенному домену доступа к серверу, на котором будет запущено ПО, необходимое для обработки запросов к хранимым файлам (вэб-сервер). Как правило, в услугу входит предоставление места для постовой корреспонденции, баз данных, DNS, файлового хранилища на специально выделенном файл-сервере и т.п., а также поддержка функционирования соответствующих серверов.

Один из главных критериев выбора хостинга – операционная система, т.к. от нее зависит ПО, которое будет поддерживать функциональность сервисов.

Прочие критерии:

- поддержка CGI (стандарт интерфейса, используемого внешней программой для связи с веб-сервисом, или по-другому «скрипты»): Python, Ruby, PHP, Perl, ASP, JSP, Java.
- поддержка .htaccess / .htpasswd – для HTTP-сервиса Apache
- поддержка баз данных, а т.ж. установленные модули и фреймворки для каждой из возможностей

Как услуга хостинг имеет такие критерии, как:

- размер дискового пространства под файлы пользователя (память)
- количество месячного трафика
- количество сайтов, которые можно разместить с одной учетки
- кол-во FTP пользователей
- кол-во e-mail ящиков, пространство под почту
- кол-во баз данных и пространство под них
- кол-во одновременных процессов на пользователя
- кол-во ОЗУ и максимальное время исполнения на каждый процесс пользователя

Качественные показатели:

- свободные ресурсы CPU (центральный процессор), оперативной памяти, которые влияют на быстродействие сервера
- пропускная способность каналов, от которой зависит загрузка информации
- удаленность оборудования хостера от ЦА, которая влияет на скорость загрузки информации

Виды хостинга:

- виртуальный хостинг – сервер с множеством сайтов, владельцы которых имеют одинаковые права и обязанности
- виртуальный выделенный сервер (VPS/VDS) – автономная (выделенная часть дискового пространства на сервере и фиксированные ресурсы. Владелец получает права админа и может сам устанавливать и настраивать программы)
- выделенный сервер - полное владение сервером с отдельной ОС, ПО
- colocation – размещение сервера, которым владеет отдельный человек или компания, в data-центре хостинговой компании.

Сервер ([аппаратное обеспечение](#))

Это выделенный или специализированный компьютер (ПК или рабочая станция) для выполнения сервисного ПО (в т.ч. серверов тех или иных задач).

Это компьютер, выполняющий серверные задачи, или компьютер (или иное аппаратное обеспечение), специализированный (по форм-фактору и/или ресурсам) для использования в качестве аппаратной базы для серверов услуг (иногда — услуг определённого направления), разделяя ресурсы компьютера с программами, запускаемыми пользователем. Такой режим работы называется «невыделенным», в отличие от «выделенного» (англ. dedicated), когда компьютер выполняет только сервисные функции. Строго говоря, на рабочей станции (для примера, под управлением Windows XP) и без того всегда работает несколько серверов — сервер удалённого доступа (терминальный сервер), сервер удалённого доступа к файловой системе и системе печати и прочие удалённые и внутренние серверы.

Сервер ([ПО](#))

Это программный компонент вычислительной системы, выполняющий сервисные функции по запросу клиента, предоставляя ему доступ к определенным ресурсам и услугам.

Для взаимодействия с клиентом (-ами) сервер выделяет необходимые ресурсы межпроцессного взаимодействия и ожидает запросы на открытие соединения. В зависимости от процесса сервер может обслуживать запросы одной системы или на других машинах через каналы передачи данных (пр.: COM-порт) или сетевые соединения.

Формат запросов клиента и ответов сервера определяется протоколом. Спецификации открытых протоколов описываются открытыми стандартами (пр.: документы RFC определяют протоколы Интернета).

Классификация стандартных серверов по типу услуг:

- Универсальные – не выполняют услуг самостоятельно, они предоставляют серверам услуг упрощенный интерфейс к ресурсам межпроцессного взаимодействия и/или унифицированный доступ клиентов к услугам.
 - inetd (internet super-server daemon – демон сервисов IP) – стандартное средство UNIX-систем – программа, позволяющая писать серверы TCP/IP (и сетевых протоколов других семейств), работающие с клиентом через перенаправленные inetd потоки стандартного ввода и вывода (stdin и stdout).
 - RPC (Remote Procedure Call – удаленный вызов процедур) – система интеграции серверов в виде процедур, доступных для вызова удаленным пользователем через унифицированный интерфейс. Сейчас в большинстве UNIX-систем и Windows используется унифицированный интерфейс от Sun Microsystems, изобретенный для их ОС (SunOS, Solaris, Unix-система).
 - Прикладные клиент-серверные технологии Windows:
 - (D-)COM – (Distributed) Component Object Model – модель составных объектов – позволяет одним программам совершать операции над объектами данных, используя процедуры других программ. Изначально предназначена для внедрения и связывания объектов (OLE – Object Linking and Embedding), но в общем позволяет писать широкий спектр различных прикладных серверов. COM – работает в пределах одного компьютера, DCOM – доступна удаленно через RPC.
 - Active-X – расширение COM и DCOM для создания мультимедийных приложений.

Универсальные серверы часто используются для написания информационных серверов – тех, что не нуждаются в специфической работе с сетью и не имеющих никаких задач, кроме обслуживания клиентов (пр.: обычные консольные программы и скрипты могут выступать в роли серверов для inetd).

Большинство внутренних и сетевых специфических серверов Windows работают через универсальные серверы (RPC, (D-)Com).

- Маршрутизация – не совсем сервер, скорее базовая функция поддержки сети операционной системой.

Для TCP/IP маршрутизация является базовой функцией стека IP (кода поддержки TCP/IP). Каждая система в сети выполняет маршрутизацию своих пакетов к месту назначения. Маршрутизаторы (роутеры или шлюзы) выполняют маршрутизацию чужих пакетов (форвардинг), управляемые через таблицы маршрутов и правила, их задачи:

- принять пакет
- найти машину, которой предназначался пакет, или следующий в цепи маршрутизатор
- передать пакет или вернуть ICMP-сообщение о невозможности доставки пакета по причинам:
 - назначение недостижимо (destination unreachable) – у пакета кончилось «время жизни» прежде, чем он был доставлен
 - хост недостижим (host un.) – компьютер или сл. маршрутизатор выключен или не существует

- сеть недостижима (network un.) – маршрутизатор не имеет маршрута в сеть назначения.
 - если пакет не может быть доставлен по причине чрезмерной загрузки маршрутизатора или сети – отбросить пакет без уведомлений.
- Динамическая маршрутизация – имеет своей задачей сбор информации о текущем состоянии сложной сети и поддержание таблицы маршрутов через эту сеть, чтобы обеспечить доставку пакета по кратчайшему и самом эффективному маршруту.

Из решений динамической маршрутизации клиент-серверную модель использует только BGP (Border Gateway Protocol – протокол пограничного шлюза), применяемый для глобальной маршрутизации.

Локальные решения (RIP (протокол маршрутной информации – простейший протокол, позволяющий маршрутизаторам динамически обновлять маршрутную информацию по данным от соседних маршрутизаторов небольшой сети), OSPF (ПДМ. позволяющий отслеживать состояние канала и использующий для нахождения кратчайшего пути алгоритм Дейкстры)) используют в своей работе бродкастовые (широковещательный канал - поток данных предназначен для всех участников сети) и мультикастовые (мультивещание, много адресное – адрес сетевого назначения – мультикастная группа) рассылки.

- Сетевые службы – обеспечивают функционирование сети (пр.: серверы DHCP (прикладной протокол, позволяющий сетевым устройствам автоматически получать IP-адрес и др. параметры, необходимые для работы в сети TCP/IP) и BOOTP (прикладной протокол, позволяющий клиенту автоматически получать IP-адрес, обычно при загрузке компьютера) обеспечивают стартовую инициализацию серверов и рабочих станций, а DNS – трансляцию имен в адреса и наоборот).

Серверы туннелирования (пр.: VPN) и прокси-серверы обеспечивают связь с сетью, недоступной роутингом.

Серверы AAA и Radius обеспечивают в сети единую аутентификацию, авторизацию и ведение логов доступа.

- Информационные службы – к ним относятся как простейшие серверы, сообщающие информацию о хосте (time, daytime, motd) и пользователях (finger, ident (протокол, описывающий способ идентификации пользователя для конкретного соединения TCP)), так и серверы для мониторинга (пр.: SNMP – стандартный интернет-протокол для управления устройствами в IP-сетях на основе архитектур TCP/UDP; поддерживаемые устройства – маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, модемные стойки и др.). Большинство информационных служб работают через универсальные серверы.

Особый вид информ. служб – сервисы синхронизации времени – NTP (сетевой протокол, используемый для синхронизации часов на компьютере с помощью сетей с переменной латентностью). Помимо установки времени на компьютере, NTP периодически опрашивает другие серверы для сверки точности собственного времени, а т.ж. скорости хода часов путем замедления или ускорения.

- Файловые серверы – обеспечивают доступ к файлам на диске сервера. Прежде всего они передают файлы по протоколам:
 - FTP -
 - TFTP – используется в основном при первоначальной загрузке бездисковых рабочих станций, в отличие от FTP не содержит возможности аутентификации и основан на транспортном протоколе UDP.
 - SFTP – протокол прикладного уровня, предназначенный для копирования и выполнения других операций с файлами поверх надежного и безопасного соединения.

- HTTP – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML (текстовые данные), в наст. вр. используется для передачи произвольных данных (веб-страницы, картинки, музыка и т.д.).

Другие серверы позволяют монтировать дисковые разделы сервера в дисковое пространство клиента и полноценно работать с файлами на них. Например, серверы протоколов NFS (протокол сетевого доступа к файловым системам, который позволяет подключать удаленные файловые системы через сеть) и SMB (сетевой протокол прикладного уровня для удаленного доступа к файлам, принтерам и др. сетевым ресурсам, а т.ж. межпроцессного взаимодействия), которые работают через интерфейс RPC.

Недостатки файл-серверной системы:

- большая нагрузка на сеть, высокие требования к пропускной способности (делает практически невозможной одновременную работу большого числа пользователей с большим объемом данных)
 - обработка данных осуществляется на компьютере пользователя (требования к уровню их аппаратного обеспечения)
 - блокировка данных при работе с ними одним пользователем делает невозможной работу с ними других пользователей
 - безопасность, т.к. для работы с файлом придется дать пользователю полный доступ к нему, когда необходима всего часть файла
- Серверы доступа к данным – обслуживают БД и отдают данные по запросу. Один из простейших – LDAP (Lightweight Directory Access Protocol – облегченный протокол доступа к спискам).
- Для БД единого протокола нет, однако ряд БД отъединяет использование единых правил формирования запросов – языка SQL (Structured Query Language – язык структурированных запросов). Остальные БД – NoSQL.
- Медиа серверы – предоставляют сети доступ к мультимедийным источникам, от аудио/видео по запросу до стриминга в аудио/видео в реальном времени.

- VoIP/ IP-телефония – программные коммутаторы (софтсвитчи), IP-АТС (автоматическая телефонная станция на основе межсетевого протокола IP операторского уровня), виртуальные АТС и серверы ВКС (сервер многочастотной конференции для неск. польз.), а также специализированные серверы Интернет-сервисов (пр.: Skype) обеспечивают пользователей возможностью голосовой и видеосвязи в реальном времени посредством компьютерной сети. Кроме потоковой передачи медиа данных, сервер IP-телефонии подобно классической АТС реализует возможность регистрации окончного терминала, маршрутизацию вызова и корректное установление соединения между пользователями и др.

В отдельных случаях, в зависимости от реализуемой технологии и административных настроек, VoIP-сервер может обеспечивать только управление — регистрацию пользователя в сети и коммутацию поступающих вызовов, без непосредственного участия в передаче медиа-данных между клиентскими терминалами. В этом случае потоковые данные с полезной нагрузкой передаются напрямую между конечными пользователями (peer-to-peer) и / или некоторыми промежуточными устройствами, приложениями. Известно, что такой вариант прямой связи с управлением через сервер применяется в Skype, Viber, Telegram и WhatsApp. Также, подобный режим нередко применяется в корпоративных IP-АТС.

В качестве клиентских терминалов к VoIP-серверу могут выступать VoIP-телефоны, видеотелефоны, программные телефоны (софтфоны), а также обычные аналоговые телефонные аппараты подключенные через VoIP-шлюз. Сервер IP-телефонии может работать как самостоятельное устройство для обеспечения связи между внутренними пользователями или быть подключенным к какой-либо сторонней сети, в том числе к телефонной сети общего пользования, через Интернет или через сеть оператора телефонной связи.

- Службы обмена сообщениями – эл. почты, работающие по протоколу SMTP. SMTP-сервер принимает сообщение и доставляет его в локальный почтовый ящик пользователя или на другой SMTP-сервер (сервер назначения или промежуточный). На многопользовательских компьютерах пользователи работают с почтой прямо на терминале или веб-интерфейсе. Для работы с почтой на ПК почта забирается из почтового ящика через серверы, работающие по протоколам POP3 и IMAP. Для организации конференций используются серверы новостей, работающие по протоколу NNTP. Для обмена сообщениями в реальном времени существуют серверы чатов по многочисленным протоколам (пр.: IRC, Jabber (XMPP), OSCAR).
- Серверы удаленного доступа – через соответствующую клиентскую программу обеспечивают пользователя аналогом локального терминала (текстового или графического) для работы на удаленной системе.

Для обеспечения доступа к командной строке служат серверы telnet, RSH, SSH.

В Unix-системах есть встроенный сервер удаленного доступа к графическому интерфейсу – X Window System. В Windows – терминальный сервер.

SNMP-протокол позволяет удаленно производить мониторинг и конфигурацию, для этого на ПК или АУ должен быть SNMP-сервер.

- Серверы приложений – предоставляют сети прикладные сервисы.
- Игровые серверы - служат для одновременной игры нескольких пользователей в единой игровой ситуации. Некоторые игры имеют сервер в основной поставке и позволяют запускать его в невыделенном режиме (то есть позволяют играть на машине, на которой запущен сервер)
- Прочие серверы:
 - Принт-серверы позволяют пользователям сети совместно использовать общий принтер.
 - Факс-сервер позволяет пользователям сети отправлять факсимильные сообщения.

Серверные решения – ОС и/или пакеты программ, оптимизированные под выполнение компьютером функций сервера и/или содержание в своем составе комплект программ для реализации типичного набора серверов (пр.: Unix-системы, изначально предназначенные для реализации серверной инфраструктуры).

Также необходимо выделить пакеты серверов и сопутствующих программ (например комплект веб-сервер/PHP/MySQL для быстрого развертывания хостинга) для установки под Windows (для Unix свойственна модульная или «пакетная» установка каждого компонента, поэтому такие решения редки, но они существуют. Наиболее известное — LAMP).

В интегрированных серверных решениях установка всех компонентов выполняется единовременно, все компоненты в той или иной мере тесно интегрированы и предварительно настроены друг на друга. Однако в этом случае замена одного из серверов или вторичных приложений (если их возможности не удовлетворяют потребностям) может представлять проблему.

Серверные решения служат для упрощения организации базовой ИТ-инфраструктуры компаний, то есть для оперативного построения полноценной сети в компании, в том числе и «с нуля». Компоновка отдельных серверных приложений в решение подразумевает, что решение предназначено для

выполнения большинства типичных задач; при этом значительно снижается сложность развёртывания и общая стоимость владения ИТ-инфраструктурой, построенной на таких решениях.

Клиент-сервер

Это вычислительная или сетевая архитектура (фактически ПО), в которой задания или сетевая нагрузка распределены между поставщиками услуг (серверы) и заказчиками услуг (клиентами).

Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов (но м.б. расположены и на одном компьютере).

Программы-серверы ожидают от клиентских программ запросы и предоставляют им ресурсы в виде данных (пр.: загрузка файлов через HTTP, FTP, Bit Torrent, потоковые мультимедиа, а также – работа с БД) или в виде сервисных функций (пр.: работа с эл. почтой, общение в мессенджерах, просмотр web-страниц).

В дополнение к клиент-серверной модели распределенные вычислительные приложения часто используют peer-to-peer архитектуру (одноранговая, децентрализованная или пиринговая сеть – оверлейная комп. сеть, основанная на равноправии участников; часто в ней отсутствуют выделенные серверы, а каждый узел (peer) как является клиентом, так и выполняет функции сервера. В отличие от архитектуры клиент-сервера, такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов. Участниками сети являются все узлы.

Преимущества peer-to-peer:

- нагрузка и объем данных распределяются между несколькими компьютерами
- если один из них недоступен, общедоступные файлы разделяются между другими компьютерами

Преимущества клиент-сервер:

- отсутствие дублирования кода программы-сервера программами-клиентами
- ниже требования к компьютерам-клиентам
- сервер, как правило, защищен лучше большинства клиентов
- проще сформировать иерархию доступа

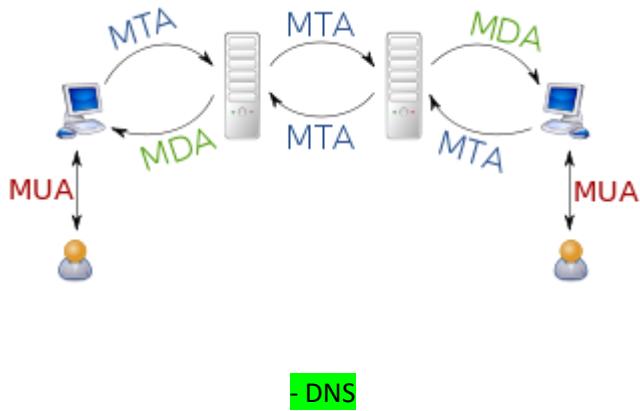
Недостатки клиент-сервер:

- неработоспособность сервера – проблема
- поддержка работы – нужен сис. админ
- дорогое железо, высокие требования к оборудованию, которые растут по мере прогнозируемого роста нагрузки

Многоуровневая архитектура «клиент — сервер» — разновидность архитектуры «клиент — сервер», в которой функция обработки данных вынесена на несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

Принцип работы эл. почты

Наиболее распространенные почтовые сервера: [gmail](#), [Sendmail](#), [Exim](#), [Postfix](#).



Domain Name System – система доменных имен – компьютерная распределенная система для получения информации о доменах.

Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получении информации о маршрутизации почты и/или обслуживающих узлах для протоколов в домене (SRV-запись).

Распределенная БД DNS поддерживается с помощью иерархии DNS-серверов, взаимодействующих по определенному протоколу.

Основа DNS – представление о иерархической структуре доменных имен и зонах. Каждый сервер, отвечающий за имя, может передать ответственность за дальнейшую часть домена другому серверу (с административной т.з. – другой организации или человеку), что позволяет возложить ответственность за актуальность информации на серверы различных организаций (людей), отвечающих только за «свою» часть доменного имени.

DNS Security Extensions (DNSSEC) – средства проверки целостности передаваемых данных.

DANE – набор спецификаций IETF, обеспечивающий аутентификацию объектов адресации (сертификатов, обеспечивающих безопасное и защищенное соединение транспортного и прикладного уровней) и предоставляемых сервисов с помощью DNS.

Ключевые характеристики DNS:

- Распределенность администрирования – ответственность за разные части иерархической структуры несут разные люди и организации.
- Распределенность хранения информации – каждый узел сети обязан хранить только те данные, которые входят в его зону ответственности, и (возможно) адреса корневых DNS-серверов.
- Кэширование информации – узел может хранить некоторое количество данных не из своей зоны ответственности для уменьшения нагрузки на сеть.
- Иерархическая структура – все узлы объединены в дерево, каждый узел может или самостоятельно определять работу нижестоящих узлов, или делегировать их другим узлам.
- Резервирование – за хранение и обслуживание своих узлов (зон) отвечают (обычно) несколько серверов, разделенные как физически, так и логически, что обеспечивает сохранность данных и продолжение работы даже в случае сбоя в одном из узлов.

Дополнительные возможности:

- поддержка динамических обновлений
- защита данных (DNSSEC) и транзакций (TSIG)
- поддержка различных типов информации

Терминология:

- Домен – узел в дереве имен вместе со всеми подчиненными ему узлами, т.е. именованная ветвь или поддерево в дереве имен. Его структура отражает порядок следования узлов в иерархии,

читая слева направо от младших доменов до доменов высшего уровня: вверху – корневой домен (имеет идентификатор «.», обслуживается корневыми серверами DNS), ниже – домены первого уровня (доменные зоны), затем – второго уровня и т.д. DNS позволяет не указывать точку корневого домена (в конце).

- Поддомен – подчиненный домен, является частью домена более высокого уровня.
- Ресурсная запись – единица хранения и передачи информации в DNS. Имеет имя (т.е. привязана к определенному доменному имени), тип и поле данных, формат и содержание которого зависят от типа.
- Зона – часть дерева доменных имен (включая ресурсные записи), размещаемая как единое целое на некотором сервере доменных имен (DNS-сервере), чаще – одновременно на нескольких серверах. Цель отделения – передача ответственности за соответствующий домен другому лицу или организации (делегирование). Являясь связной частью дерева, зона тоже является деревом с дочерними зонами.
- Делегирование – операция передачи ответственности за часть дерева доменных имен отдельному лицу или организации. Технически это выделение части дерева в отдельную зону и ее размещение на DNS-сервере под управлением лица или организации. При этом в родительскую зону включаются «склеивающие» ресурсные записи (NS и A), содержащие указатели на DNS-сервера дочерней записи.
- DNS-сервер – специализированное ПО для обслуживания DNS, а т.ж. компьютер, на котором оно выполняется. М.б. ответственным за некоторые зоны и/или может перенаправлять запросы вышестоящим серверам.
- DNS-клиент – специализированная библиотека (или программа) для работы с DNS (DNS-сервер может выступать в этой роли).
- Авторитетность – признак размещения зоны на DNS-сервере. Ответы DNS-сервера м.б. двух типов: авторитетные (сервер заявляет, что сам отвечает за зону) и неавторитетные (сервер обрабатывает запрос и возвращает ответ других серверов). Иногда вместо передачи запроса дальше DNS-сервер может вернуть уже известное ему значение (режим кэширования).
- DNS-запрос – DNS query – запрос от клиента (или сервера) к серверу. М.б. рекурсивным или не рекурсивным.

Принцип работы DNS:

Имя не тождественно IP-адресу: один IP-адрес может иметь несколько имен, что позволяет поддерживать на одном компьютере несколько веб-сайтов (виртуальный хостинг). Но и имя может иметь множество IP-адресов, что позволяет создавать балансировку нагрузки.

Для повышения устойчивости системы используется множество серверов, содержащих идентичную информацию, а в протоколе есть средства, позволяющие поддерживать синхронность информации, расположенной на разных серверах. Существует 13 корневых серверов, их адреса практически не изменяются.

Протокол DNS использует для работы TCP- или UDP-порт 53 для ответов на запросы. Традиционно запросы и ответы отправляются в виде одной UDP-датаграммы. TCP используется, когда размер данных ответа превышает 512 байт, и для AXFR-запросов.

Рекурсия:

В DNS это алгоритм поведения DNS-сервера: выполнить от имени клиента полный поиск нужной информации по всей системе DNS, при необходимости обращаясь к другим DNS-серверам.

Рекурсивный DNS-запрос требует полного поиска, нерекурсивный (итеративный) – нет.

Сам DNS-сервер м.б. рекурсивным (умеющим выполнять полный поиск) и нерекурсивным. Некоторые программы DNS-серверов, например, BIND, можно сконфигурировать так, чтобы запросы одних клиентов выполнялись рекурсивно, а запросы других — нерекурсивно.

При нерекурсивном запросе, неумении или запрете сервера выполнять рекурсивные запросы, он либо возвращает данные о зоне, за которую ответственен, либо ошибку.

Нерекурсивный сервер, выдающий информацию о серверах с большим кол-вом информации, м.б. использован для DoS-атак.

Как работает:

Предположим, мы набрали в браузере адрес ru.wikipedia.org. Браузер ищет соответствие этого адреса IP-адресу в файле hosts. Если файл не содержит соответствия, то далее браузер спрашивает у сервера DNS: «какой IP-адрес у ru.wikipedia.org»? Однако сервер DNS может ничего не знать не только о запрошенном имени, но и даже обо всём домене wikipedia.org. В этом случае сервер обращается к корневому серверу — например, 198.41.0.4. Этот сервер сообщает — «У меня нет информации о данном адресе, но я знаю, что 204.74.112.1 является ответственным за зону org.» Тогда сервер DNS направляет свой запрос к 204.74.112.1, но тот отвечает «У меня нет информации о данном сервере, но я знаю, что 207.142.131.234 является ответственным за зону wikipedia.org.» Наконец, тот же запрос отправляется к третьему DNS-серверу и получает ответ — IP-адрес, который и передаётся клиенту — браузеру.

В данном случае при разрешении имени, то есть в процессе поиска IP по имени:

- браузер отправил известному ему DNS-серверу рекурсивный запрос — в ответ на такой тип запроса сервер обязан вернуть «готовый результат», то есть IP-адрес, либо пустой ответ и код ошибки NXDOMAIN;
- DNS-сервер, получивший запрос от браузера, последовательно отправлял нерекурсивные запросы, на которые получал от других DNS-серверов ответы, пока не получил ответ от сервера, ответственного за запрошенную зону;
- остальные упоминавшиеся DNS-серверы обрабатывали запросы нерекурсивно (и, скорее всего, не стали бы обрабатывать запросы рекурсивно, даже если бы такое требование стояло в запросе).

DNS-сервер браузера кэширует результат и в следующий раз не производятся опросы прочих серверов.

Обратный DNS-запрос:

С записью DNS могут быть сопоставлены различные данные, в том числе и какое-либо символьное имя. Существует специальный домен in-addr.arpa, записи в котором используются для преобразования IP-адресов в символьные имена. Например, для получения DNS-имени для адреса 11.22.33.44 можно запросить у DNS-сервера запись 44.33.22.11.in-addr.arpa, и тот вернёт соответствующее символьное имя. Обратный порядок записи частей IP-адреса объясняется тем, что в IP-адресах старшие биты расположены в начале, а в символьных DNS-именах старшие (находящиеся ближе к корню) части расположены в конце.

Записи DNS, или ресурсные записи (resource records, RR), — единицы хранения и передачи информации в DNS. Каждая ресурсная запись состоит из следующих полей:

- имя (NAME) — доменное имя, к которому привязана или которому «принадлежит» данная ресурсная запись,
- тип (TYPE) ресурсной записи — определяет формат и назначение данной ресурсной записи,
- класс (CLASS) ресурсной записи; теоретически считается, что DNS может использоваться не только с TCP/IP, но и с другими типами сетей, код в поле класс определяет тип сети,
- TTL (Time To Live) — допустимое время хранения данной ресурсной записи в кэше неответственного DNS-сервера,
- длина поля данных (RDLEN),
- поле данных (RDATA), формат и содержание которого зависит от типа записи.

Наиболее важные типы DNS-записей:

- Запись A (address record) или запись адреса связывает имя хоста с адресом протокола IPv4. Например, запрос A-записи на имя referrals.icann.org вернёт его IPv4-адрес — 192.0.34.164.
- Запись AAAA (IPv6 address record) связывает имя хоста с адресом протокола IPv6. Например, запрос AAAA-записи на имя K.ROOT-SERVERS.NET вернёт его IPv6-адрес — 2001:7fd::1.
- Запись CNAME (canonical name record) или каноническая запись имени (псевдоним) используется для перенаправления на другое имя.
- Запись MX (mail exchange) или почтовый обменник указывает сервер(ы) обмена почтой для данного домена.
- Запись NS (name server) указывает на DNS-сервер для данного домена.
- Запись PTR (pointer) обратная DNS-запись или запись указателя связывает IP-адрес хоста с его каноническим именем. Запрос в домене in-addr.arpa на IP-адрес хоста в reverse-форме вернёт имя (FQDN) данного хоста. Например, для IP-адреса 192.0.34.164 запрос записи PTR 164.34.0.192.in-addr.arpa вернёт его каноническое имя referrals.icann.org. В целях уменьшения объёма спама многие серверы-получатели электронной почты могут проверять наличие PTR-записи для хоста, с которого происходит отправка. В этом случае PTR-запись для IP-адреса должна соответствовать имени отправляющего почтового сервера, которым он представляется в процессе SMTP-сессии.
- Запись SOA (Start of Authority) или начальная запись зоны указывает, на каком сервере хранится эталонная информация о данном домене, содержит контактную информацию лица, ответственного за данную зону, тайминги (параметры времени) кеширования зонной информации и взаимодействия DNS-серверов.
- SRV-запись (server selection) указывает на серверы для сервисов, используется, в частности, для Jabber (XMPP) и Active Directory.

Зарезервированные доменные имена – [здесь](#). Можно использовать в кач-ве примеров в документации.

Доменное имя может состоять только из ограниченного набора ASCII-символов, позволяя набрать адрес домена независимо от языка пользователя. ICANN утвердил основанную на Punycode систему IDNA, преобразующую любую строку в кодировке Unicode в допустимый DNS набор символов.

Серверы имен:

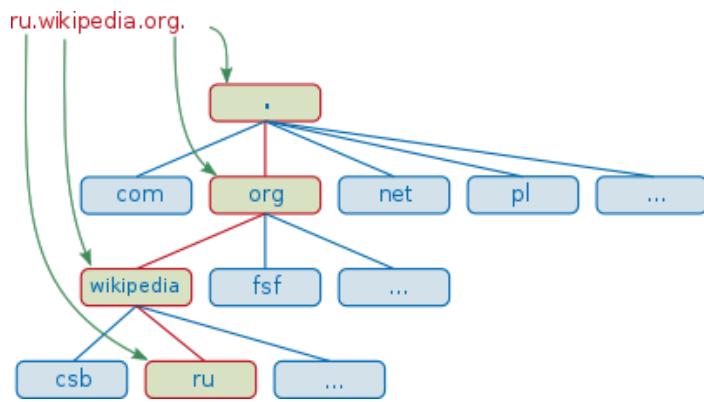
- [BIND](#) (Berkeley Internet Name Domain)
- [djbdns](#) ([Daniel J. Bernstein's DNS](#))
- [Dnsmasq](#)
- [MaraDNS](#)
- [NSD](#) (Name Server Daemon)
- [PowerDNS](#)
- [OpenDNS](#)
- [Microsoft DNS Server](#) (в серверных версиях операционных систем [Windows NT](#))
- [MyDNS](#)

- HTTP

- браузеры

- доменные имена

Пример структуры доменного имени



- архитектура сети и место бэка в ней

2. Знать особенности Python, где и как применяется в бэке, почему именно он, плюсы и минусы

4. Редактор IDE (PyCharm, VS Code, Visual Studio, IntelliJ Idea):

-какие есть

- почему PyCharm, его + и -

5. Терминал (PowerShell, Bash, Zsh) - то же, что и про редактор

6. Фреймворк (Django): плюсы и минусы, на что способен.

7. Базы данных:

- реляционные (PostgreSQL, MySQL, MS SQL, Azure Cloud SQL)

- нереляционные (NoSQL)(MongoDB, Redis, Cassandra, AWS, Firebase)

8. API (RESTful Api, Swagger)

Программный интерфейс приложения – это описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой.

Обычно входит в описание к-л интернет-протокола (пр.: RFC – Recall For Comments, заявка, содержит спецификации и стандарты, широко применяемые во всемирной сети), программного класса (фреймворка) или стандарта вызова функций ОС.

Часто реализуется отдельной программной библиотекой или сервисом ОС.

Облегчает процесс написания приложений, так как содержит набор действий, исключающих необходимость углубленного знания ряда областей. Так, например, разработчику не нужно знать операций, которые происходят в глубинах файловой системы, для того чтобы копировать файл, так как API предоставит функцию для данного действия.

Если рассмотреть программу (библиотеку, модуль) как черный ящик, API поможет им управлять.

Программный компоненты иерархично взаимодействуют посредством API (высокоуровневые компоненты используют API низкоуровневых и т.д. по нисходящей).

По такому принципу построены протоколы передачи данных по интернету. Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью нижестоящего уровня и предоставляет функциональность вышестоящему.

Понятие протокола близко по смыслу к понятию API: оба являются абстракцией функциональности, только первое связано с передачей данных, а второе – со взаимодействием приложений.

API библиотеки функций и классов включают в себя описание сигнатур и семантики функций.

- Сигнатура функции – часть общего объявления функции, позволяющая средствам трансляции идентифицировать функцию среди других.

Т.к. в разных языках разные представления о сигнатуре функций, есть возможность пользоваться перезагрузкой функции, т.е. использовать функции с одинаковыми названиями.

Также различают сигнатуру вызова (составляется по синтаксической конструкции вызова функции с учетом ее области видимости, ее имени, последовательности фактических типов аргументов в вызове и типа результата) и сигнатуру реализации функции (в ней участвуют некоторые элементы из синтаксической конструкции объявления функции: спецификатор области видимости функции, ее имя и последовательность формальных типов аргументов).

- Семантика функции – описание того, что эта функция делает. Включает в себя описание того, что является результатом вычисления функции, как и от чего этот результат зависит. Он может зависеть от аргументов, а иногда и от состояния. Логика этих зависимостей и изменений относится к семантике функции.

У каждой ОС имеют API, которое позволяет создавать приложения для данной ОС. Главный API ОС – множество системных вызовов (обращений прикладной программы к ядру ОС для выполнения к-л операции).

Единые стандарты API внутри одной ОС гарантируют правильную и схожую работу программ, написанных в рамках этого API.

Однако различия API разных ОС приводят к затруднению переноса приложений между платформами. Возможные пути решения:

- написание «промежуточных» API (API графических интерфейсов wxWidgets, GTK и т. п.)
- написание библиотек, которые отображают системные вызовы одной ОС в системные вызовы другой (Wine, cygwin и т. п.)
- введение стандартов кодирования в языках программирования (пр.: стандартная библиотека языка C)
- написание интерпретируемых языков, реализуемых на разных платформах (sh, Python, Perl, PHP, Tcl, JavaScript, Ruby и т. д.)

Также в распоряжении программиста часто находится несколько различных API, позволяющих добиться одного и того же результата.

Основными сложностями существующих многоуровневых систем API, таким образом, являются:

- Сложность портирования программного кода с одной системы API на другую (например, при смене ОС);
- Потеря функциональности при переходе с более низкого уровня на более высокий. Грубо говоря, каждый «слой» API создаётся для облегчения выполнения некоторого стандартного набора

операций. Но при этом реально затрудняется либо становится принципиально невозможным выполнение некоторых других операций, которые предоставляет более низкий уровень API.

Наиболее известные API:

- ОС: [Amiga ROM Kernel](#), [Cocoa](#), [Linux Kernel API](#), [OS/2 API](#), [POSIX](#), [Windows API](#)
- Графических интерфейсов: [DirectDraw](#)/[Direct3D](#) (часть [DirectX](#)), [GDI](#), [GDI+](#), [GTK](#), [SFML](#), [Motif](#), [OpenGL](#), [OpenVG](#), [Qt](#), [SDL](#), [Vulkan](#), [Tk](#), [wxWidgets](#), [X11](#), [Zune](#)
- Звуковых интерфейсов: [DirectMusic](#)/[DirectSound](#) (часть [DirectX](#)), [OpenAL](#)
- Аутентификационных систем: [BioAPI](#), [PAM](#)

Web API – используется в веб-разработке, содержит определенный набор HTTP-запросов, а т.ж. определение структуры HTTP-ответов, для выражения которых используют XML – или JSON-формат.

Сейчас перешли от SOAP к REST типу коммуникации.

Веб-интерфейсы, обеспечивающие сочетание нескольких сервисов в новых приложениях, известны как гибридные.

Структуры данных

Статический массив

Динамический массив

Ассоциативный массив (HASHTABLE)

9. Аутентификация (OAuth 2.0, JWT):

- Провайдеры (Auth0, Firebase)

10. Паттерны проектирования (чит. кн. “Банда 4x”):

- порождающие

- структурные

- поведенческие

11. Тесты (+изучить фреймворки для написания этих тестов):

- Unit

- интеграционные

12. Доп. инструменты разработки

- менеджеры пакетов (NuGet, npm, yarn)

- системы контроля версий (git, GitHub, TFS, BitBucket)

Также:

- умение декомпозировать предметную область (понять уже написанный код, легко в нем ориентироваться)
- знание вэб-сервисов, умение их создавать
- сетевые протоколы (как открыть порт, к нему приконнектиться, открыть приложение вызовов)
- ui фреймфорки

Unix-подобная ОС

Это свободные/открытые ОС, созданные по подобию Unix (пр.: Linux, FreeBSD).

Операционные системы семейства Unix характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой, взаимодействие осуществляется через единую файловую систему, а для работы с утилитами используется командная оболочка.

Межпроцессное взаимодействие (англ. inter-process communication, IPC)

Это обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

Из механизмов, предоставляемых ОС и используемых для IPC, можно выделить:

- механизмы обмена сообщениями;
- механизмы синхронизации;
- механизмы разделения памяти;
- механизмы удалённых вызовов (RPC).

Для оценки производительности различных механизмов IPC используют следующие параметры:

- пропускная способность (количество сообщений в единицу времени, которое ядро ОС или процесс способно обработать);
- задержки (время между отправкой сообщения одним потоком и его получением другим потоком).

Именованный канал

В программировании именованный канал или именованный конвейер (англ. named pipe) — один из методов межпроцессного взаимодействия, расширение понятия конвейера в Unix и подобных ОС.

Именованный канал позволяет различным процессам обмениваться данными, даже если программы, выполняющиеся в этих процессах, изначально не были написаны для взаимодействия с другими программами.

Это понятие также существует и в Microsoft Windows, хотя там его семантика существенно отличается. Традиционный канал — «безымянен», потому что существует анонимно и только во время выполнения процесса.

Именованный канал — существует в системе и после завершения процесса. Он должен быть «отсоединен» или удалён, когда уже не используется. Процессы обычно подсоединяются к каналу для осуществления взаимодействия между ними.

Сокет (программный интерфейс)

Это название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. [Сокет](#) — абстрактный объект, представляющий конечную точку соединения.

Следует различать клиентские и серверные сокеты. Клиентские сокеты грубо можно сравнить с конечными аппаратами телефонной сети, а серверные — с коммутаторами. Клиентское приложение (пр.: браузер) использует только клиентские сокеты, а серверное (пр.: веб-сервер, которому браузер посыпает запросы) — как клиентские, так и серверные сокеты.

Майнфрейм (также [майнфрейм](#), от англ. mainframe) — большой универсальный высокопроизводительный отказоустойчивый сервер со значительными ресурсами ввода-вывода, большим объёмом оперативной и внешней памяти, предназначенный для использования в критически важных системах (англ. mission-critical) с интенсивной пакетной и оперативной транзакционной обработкой.

Файловый сервер

Выделенный сервер, предназначенный для файловых операций ввода-вывода и хранящий файлы любого типа. Как правило, обладает большим объемом дискового пространства, реализованным в формате RAID-массива для обеспечения бесперебойной работы и повышенной скорости записи и чтения данных. Есть также [файл-серверные приложения](#).

Двоичный интерфейс приложений

Он же [Application Binary Interface](#) (ABI) — набор соглашений для доступа приложения к ОС и другим низкоуровневым сервисам, спроектированный для переносимости исполняемого кода между машинами, имеющими совместимые ABI.

В отличие от API, который регламентирует совместимость на уровне исходного кода, ABI можно рассматривать как совокупность правил, позволяющих компоновщику объединять откомпилированные модули компонента без перекомпиляции всего кода, в то же время определяя двоичный интерфейс.

Двоичный интерфейс регламентирует:

- использование регистров процессора
- состав и формат системных вызовов и вызова одного модуля другим
- формат передачи аргументов и возвращаемого значения при вызове функции

Двоичный интерфейс приложений описывает функциональность, предоставляемую рядом ОС и архитектурой набора команд (без привилегированных команд).

Если интерфейс программирования разных платформ совпадает (API), код для этих платформ можно компилировать без изменений.

Если совпадают и API, и ABI, исполняемые файлы можно переносить на эти платформы без изменений.

Если API или ABI платформ различаются, код требует изменений и повторной компиляции.

API не обеспечивает совместимости среды исполнения программы — это задача двоичного интерфейса.

Есть также бинарный интерфейс встраиваемых приложений (Embedded ABI, EABI) — набор соглашений для использования во встраиваемом ПО, описывающий:

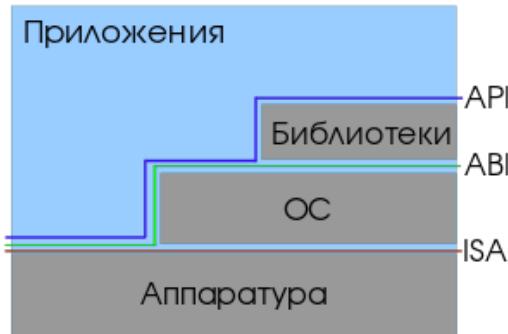
- [форматы файлов](#)

- [типы данных](#)
- способы использования регистров
- организацию [стека](#)
- соглашение о вызове функции.

Если объектный файл был создан компилятором, поддерживающим EABI, становится возможной компоновка этого объектного файла любым компоновщиком, поддерживающим тот же EABI.

Основное отличие EABI от ABI в ОС общего назначения заключается в том, что в коде приложения допускаются привилегированные команды, а динамическое связывание (компоновка) не требуется (а иногда и полностью запрещена), а также, в целях экономии памяти, используется более компактная организация стека.

Уровни и интерфейсы между ними. API, ABI и архитектура набора команд (ISA)



IP-адрес – [Internet Protocol](#) – уникальный числовой идентификатор устройства в компьютерной сети, работающий по протоколу TCP/IP. Состоит из двух частей: номера сети и номера узла.

Хост – [host](#) – любое устройство, предоставляющее сервисы формата «клиент-сервер» в режиме сервера по каким-либо интерфейсам и уникально определенное на этих интерфейсах. Глобально – любой компьютер, подключенный к локальной или глобальной сети.

Чаще всего, под «хостом» без дополнительных комментариев подразумевается хост протокола TCP/IP, то есть сетевой интерфейс устройства, подключённого к IP-сети. Как и всякий другой хост, этот имеет уникальное определение в среде сервисов TCP/IP (IP-адрес). С хостом протокола TCP/IP может быть также связана необязательная текстовая характеристика — доменное имя.

Служебная запись (SRV-запись) – стандарт в DNS, определяющий местоположение, т.е. имя хоста и номер порта серверов для определенных служб.

Некоторые Интернет-протоколы, такие как SIP и XMPP, часто требуют поддержки SRV-записей.

SRV-запись имеет такой формат:

```
_service._proto.name TTL class SRV priority weight port target
```

- service: символьное имя сервиса.
- proto: транспортный протокол, используемый сервисом, как правило TCP или UDP.
- name: доменное имя, для которого эта запись действует.
- TTL: стандарт DNS, время жизни.
- class: стандарт DNS, поле класса (это всегда IN).
- priority: приоритет целевого хоста, более низкое значение означает более предпочтительный.
- weight: относительный вес для записей с одинаковым приоритетом.

- port: Порт TCP или UDP, на котором работает сервис.
- target: канонические имя машины, предоставляющей сервис.

Распределенная БД (параллельная) – [Distributed Database](#), DDB – единая БД, составные части которой размещаются в различных узлах компьютерной сети в соответствии с к-л критерием.

Данные я-ся DDB только если они связаны в соответствии с некоторым структурным формализмом, реляционной моделью, а доступ к ним обеспечивается единым высокогорневым интерфейсом.

Могут иметь разный уровень реплицированности – от полного отсутствия дублирования информации до дублирования всей информации во всех распределенных копиях (пр.: [блокчейн](#)).

Прозрачность – распределение БД по множеству узлов невидимо для пользователей. Полная прозрачность достигается за счет: прозрачности сети, распределения, репликации, фрагментации, доступа.

Виртуальный хостинг - [shared hosting](#) – при нем множество веб-сайтов расположено на одном веб-сервере, каждый в своем разделе, по все используют одно ПО.

Существует два основных метода реализации доступа к веб-сайтам:

- по имени shared IP hosting), когда все веб-сайты используют один общий IP-адрес. Согласно протоколу HTTP/1.1, веб-браузер при запросе к веб-серверу указывает доменное имя веб-сайта в поле Host заголовка текущего запроса, и веб-сервер использует его для правильного выполнения запроса, а также копирует это имя в ячейку [HTTP_HOST] суперглобального массива \$_SERVER.
- по IP-адресу (dedicated IP hosting), при котором у каждого веб-сайта есть собственный IP-адрес, а веб-сервер имеет несколько физических или виртуальных сетевых интерфейсов.

Сертификат открытого ключа (сертификат электронной подписи, сертификат ключа подписи, сертификат ключа проверки электронной подписи) – электронный или бумажный документ, содержащий открытый ключ, информацию о владельце ключа, области применения ключа, подписанный выдавшим его Удостоверяющим центром и подтверждающий принадлежность [открытого ключа](#) владельцу.

Протокол передачи данных – [набор](#) определённых правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

TCP/IP – [набор](#) протоколов передачи данных, получивший название от двух принадлежащих ему протоколов: TCP (Transmission Control Protocol) и IP (Internet Protocol)

HTTP (Hyper Text Transfer Protocol) – это [протокол](#) передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц между компьютерами, подключёнными к одной сети.

FTP (File Transfer Protocol) – это [протокол](#) передачи файлов со специального файлового сервера на компьютер пользователя. FTP даёт возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удалённым компьютером, пользователь может скопировать файл с удалённого компьютера на свой или скопировать файл со своего компьютера на удалённый.

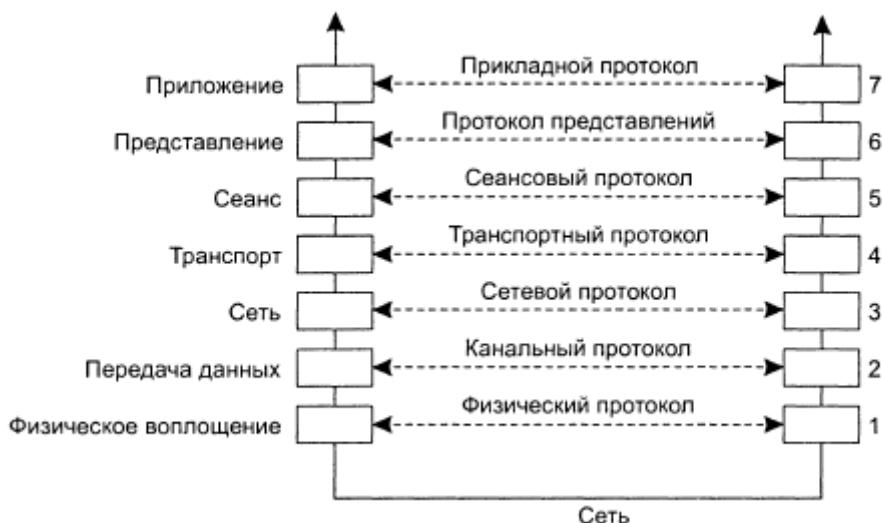
POP3 (Post Office Protocol) – это стандартный [протокол](#) почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.

SMTP (Simple Mail Transfer Protocol) — [протокол](#), который задаёт набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приёме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.

TELNET — это [протокол](#) удалённого доступа. TELNET даёт возможность абоненту работать на любой ЭВМ, находящейся с ним в одной сети, как на своей собственной, то есть запускать программы, менять режим работы и так далее. На практике возможности ограничиваются тем уровнем доступа, который задан администратором удалённой машины.

Модель OSI — 7-уровневая логическая модель работы сети. Реализуется группой протоколов и правил связи, организованных в несколько уровней:

- на физическом уровне определяются физические (механические, электрические, оптические) характеристики линий связи;
- на канальном уровне определяются правила использования физического уровня узлами сети;
- сетевой уровень отвечает за адресацию и доставку сообщений;
- транспортный уровень контролирует очерёдность прохождения компонентов сообщения;
- сессийный уровень координирует связь между двумя прикладными программами, работающими на разных рабочих станциях;
- уровень представления служит для преобразования данных из внутреннего формата компьютера в формат передачи;
- прикладной уровень является пограничным между прикладной программой и другими уровнями, обеспечивая удобный интерфейс связи для сетевых программ пользователя.



Модель — стек протоколов TCP/IP — содержит 4 уровня:

- канальный уровень (link layer),
- сетевой уровень (Internet layer),
- транспортный уровень (transport layer),
- прикладной уровень (application layer).



Демон (*daemon*) — компьютерная программа в UNIX-подобных системах, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем.

Демоны обычно запускаются во время загрузки системы. Типичные задачи демонов: серверы сетевых протоколов (HTTP, FTP, электронная почта и др.), управление оборудованием, поддержка очередей печати, управление выполнением заданий по расписанию и т. д. В техническом смысле демоном считается процесс, который не имеет управляющего терминала. Традиционно названия демон-процессов заканчиваются на букву d.

В системах Windows аналогичный класс программ называется службой (Services).

TCP/IP — сетевая [модель](#) четырехуровневой передачи цифровых данных от источника к получателю.

Каждый уровень описан протоколом передачи. На стеке протоколов передачи данных базируется Интернет.

[TCP](#) — Transmission Control Protocol — протокол управления передачей данных (пакеты – сегменты).

[IP](#) — Internet Protocol — масштабируемый протокол сетевого уровня, который объединил отдельные компьютерные сети в глобальную сеть Интернет.

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься.

Уровни:

- [Прикладной уровень](#) (Application Layer) - обеспечивает обмен данными между процессами для приложений (пр.: [HTTP](#), [RTSP](#), [FTP](#), [DNS](#));

На нем работает большинство сетевых приложений. Они имеют собственные протоколы обмена данными (пр.: [интернет браузер](#) для протокола [HTTP](#), [ftp-клиент](#) для протокола [FTP](#) (передача файлов), почтовая программа для протокола [SMTP](#) ([электронная почта](#)), [SSH](#) (безопасное соединение с удалённой машиной), [DNS](#) (преобразование символьных имён в [IP-адреса](#)) и др.)

В массе своей эти протоколы работают поверх [TCP](#) или [UDP](#) и привязаны к определённому [порту](#), например:

- [HTTP](#) на TCP-порт 80 или 8080,
- [FTP](#) на TCP-порт 20 (для передачи данных) и 21 (для управляющих команд),
- [SSH](#) на TCP-порт 22,
- запросы [DNS](#) на порт UDP (реже TCP) 53,
- обновление маршрутов по протоколу [RIP](#) на UDP-порт 520.

К этому уровню относятся: [Echo](#), [Finger](#), [Gopher](#), [HTTP](#), [HTTPS](#), [IMAP](#), [IMAPS](#), [IRC](#), [NNTP](#), [NTP](#), [POP3](#),

[POPS](#), [QOTD](#), [RTSP](#), [SNMP](#), [SSH](#), [Telnet](#), [XDMCP](#).

- [Транспортный уровень](#) (Transport Layer) - обрабатывающий связь между хостами (пр.: [TCP](#), [UDP](#), [SCTP](#), [DCCP](#); [RIP](#), протоколы маршрутизации, подобные [OSPF](#), что работают поверх [IP](#), являются частью сетевого уровня);

Могут решать проблему негарантированной доставки сообщений, а т.ж. гарантировать правильную последовательность прихода данных. В стеке TCP/IP определяют, для какого именно приложения эти данные.

Протоколы автоматической маршрутизации, логически представленные на этом уровне (поскольку работают поверх IP), на самом деле являются частью протоколов сетевого уровня; например [OSPF](#) (IP идентификатор 89).

[TCP](#) (IP идентификатор 6) — «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный [поток данных](#), дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности. В этом его главное отличие от [UDP](#).

[UDP](#) (IP идентификатор 17) протокол передачи [датаграмм](#) без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов. В приложениях, требующих гарантированной передачи данных, используется протокол [TCP](#).

И [TCP](#), и [UDP](#) используют для определения протокола верхнего уровня число, называемое [портом](#).

- [Межсетевой уровень](#) (Сетевой уровень) (Internet Layer) - обеспечивающий межсетевое взаимодействие между независимыми сетями (пр.: для TCP/IP это [IP](#) (вспомогательные протоколы, вроде [ICMP](#) и [IGMP](#), работают поверх IP, но тоже относятся к сетевому уровню; протокол [ARP](#) является самостоятельным вспомогательным протоколом, работающим поверх канального уровня);

Разработан для передачи данных из любой сети в любую сеть, независимо от протоколов нижнего уровня, а также может запрашивать данные от удалённой стороны, например в протоколе [ICMP](#) (используется для передачи диагностической информации [IP](#)-соединения) и [IGMP](#) (используется для управления [multicast](#)-потоками).

На этом уровне работают [маршрутизаторы](#), которые перенаправляют пакеты в нужную сеть путём расчёта адреса сети по [маске сети](#). Примерами такого протокола является [X.25](#) и [IPC](#) в сети [ARPANET](#).

ICMP и IGMP расположены над [IP](#) и должны попасть на следующий — транспортный — уровень, но функционально являются протоколами сетевого уровня, и поэтому их невозможно вписать в модель OSI.

Пакеты сетевого протокола [IP](#) могут содержать код, указывающий, какой именно протокол следующего уровня нужно использовать, чтобы извлечь данные из пакета. Это число — уникальный [IP-номер протокола](#). ICMP и IGMP имеют номера, соответственно, 1 и 2.

К этому уровню относятся: [DVMRP](#), [ICMP](#), [IGMP](#), [MARS](#), [PIM](#), [RIP](#), [RIP2](#), [RSVP](#)

- [Канальный уровень](#) (Network Access Layer, Link Layer) - содержащий методы связи для данных, которые остаются в пределах одного сегмента сети (пр.: [Ethernet](#), [IEEE 802.11 WLAN](#), [SLIP](#), [Token Ring](#), [ATM](#) и [MPLS](#), физическая среда и принципы кодирования информации, [T1](#), [E1](#)).

Описывает способ кодирования данных для передачи [пакета данных](#) на физическом уровне (то есть специальные последовательности бит, определяющих начало и конец пакета данных, а также обеспечивающие помехоустойчивость). [Ethernet](#), например, в полях [заголовка пакета](#) содержит указание того, какой машине или машинам в сети предназначен этот пакет.

Примеры протоколов канального уровня — [Ethernet](#), [IEEE 802.11 WLAN](#), [SLIP](#), [Token Ring](#), [ATM](#) и [MPLS](#).

[PPP](#) не совсем вписывается в такое определение, поэтому обычно описывается в виде пары протоколов [HDLC/SDLC](#).

[MPLS](#) занимает промежуточное положение между канальным и сетевым уровнем и, строго говоря, его нельзя отнести ни к одному из них.

Канальный уровень иногда разделяют на 2 подуровня — [LLC](#) и [MAC](#).

Кроме того, канальный уровень описывает среду передачи данных (будь то коаксиальный кабель, витая пара, оптическое волокно или радиоканал), физические характеристики такой среды и принцип передачи данных (разделение каналов, модуляцию, амплитуду сигналов, частоту сигналов, способ синхронизации передачи, время ожидания ответа и максимальное расстояние).

При проектировании стека протоколов на канальном уровне рассматривают помехоустойчивое кодирование — позволяющие обнаруживать и исправлять ошибки в данных вследствие воздействия шумов и помех на канал связи.

Распределение протоколов по уровням модели OSI

TCP/IP		OSI
7	Прикладной	напр., HTTP , SMTP , SNMP , FTP , Telnet , SSH , SCP , SMB , NFS , RTSP , BGP
6	Прикладной	напр., XDR , AFP , TLS , SSL
5	Сеансовый	напр., ISO 8327 / CCITT X.225 , RPC , NetBIOS , PPTP , L2TP , ASP
4	Транспортный	напр., TCP , UDP , SCTP , SPX , ATP , DCCP , GRE
3	Сетевой	напр., IP , ICMP , IGMP , CLNP , OSPF , RIP , IPX , DDP
2	Канальный	напр., Ethernet , Token ring , HDLC , PPP , X.25 , Frame relay , ISDN , ATM , SPB , MPLS , ARP
1	Физический	напр., электрические провода , радиосвязь , волоконно-оптические провода , инфракрасное излучение

Сетевая модель OSI ([The Open System Interconnection model](#)) – сетевая модель стека (магазина) сетевых протоколов OSI/ISO, посредством которой различные сетевые устройства могут взаимодействовать друг с другом; определяет различные уровни взаимодействия систем, у каждого из которых свои функции.

Протоколы связи позволяют структуре на одном хосте взаимодействовать с соответствующей структурой того же уровня на другом хосте.

На каждом уровне N два объекта обмениваются блоками данных ([PDU](#)) с помощью протокола данного уровня на соответствующих устройствах. Каждый PDU содержит блок служебных данных ([SDU](#)), связанный с верхним или нижним протоколом.

Обработка данных двумя взаимодействующими OSI-совместимыми устройствами происходит следующим образом:

1. Передаваемые данные составляются на самом верхнем уровне передающего устройства (уровень N) в протокольный блок данных (PDU).
2. PDU передается на уровень N-1, где он становится сервисным блоком данных (SDU).
3. На уровне N-1 SDU объединяется с верхним, нижним или обоими уровнями, создавая слой N-1 PDU. Затем он передается в слой N-2.
4. Процесс продолжается до достижения самого нижнего уровня, с которого данные передаются на принимающее устройство.
5. На приемном устройстве данные передаются от самого низкого уровня к самому высокому в виде серии SDU, последовательно удаляясь из верхнего или нижнего колонтитула каждого слоя до достижения самого верхнего уровня, где принимаются последние данные.

Модель					
	Уровень (layer)	Тип данных (PDU ^[14])	Функции	Примеры	Оборудование
Host layers	7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3, WebSocket	Хосты (клиенты сети), Межсетевой экран
	6. Представления (presentation)		Представление и шифрование данных	ASCII, EBCDIC, JPEG, MIDI	
	5. Сеансовый (session)		Управление сеансом связи	RPC, PAP, L2TP, gRPC	
	4. Транспортный (transport)	Сегменты (segment) / Датаграммы (datagram)	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP, Порты	
Media^[15] layers	3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk, ICMP	Маршрутизатор, Сетевой шлюз, Межсетевой экран
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация	PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.	Сетевой мост, Коммутатор, точка доступа
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	USB, RJ («витая пара», коаксиальный, оптоволоконный), радиоканал	Концентратор, Повторитель (сетевое оборудование)

Web-сервер – [сервер](#), принимающий HTTP-запросы от клиентов, обычно веб-браузеров (обозначены URL-адресами), и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Дополнительные функции:

- автоматизация работы веб-страниц
- ведение журнала обращений пользователя к ресурсам
- аутентификация и авторизация пользователей

- поддержка динамически генерируемых страниц
- поддержка HTTPS для защищенных соединений с клиентами.

65% рынка занимает web-сервер Apache – свободный веб-сервер, наиболее часто используемый в Unix-подобных ОС.

Прочие:

- [IIS](#) от компании [Microsoft](#), распространяемый с ОС семейства [Windows](#).
- [nginx](#) – свободный веб-сервер, разрабатываемый [Игорем Сысоевым](#) с 2002 года и пользующийся большой популярностью на крупных сайтах^[2] ^[3],
- [lighttpd](#) – свободный веб-сервер.
- [Google Web Server](#) – веб-сервер разработанный компанией [Google](#).
- [Resin](#) – свободный веб-сервер приложений.
- [Cherokee](#) – свободный веб-сервер, управляемый только через web-интерфейс.
- [Rootage](#) – веб-сервер, написанный на [Java](#).
- [THTTPD](#) – простой, маленький, быстрый и безопасный веб-сервер.
- [Open Server](#) – бесплатная программа с графическим интерфейсом использует множество исключительно свободного программного комплекса.
- [H2O](#) – свободный быстрый веб-сервер, написанный на [C](#).
- [nghhttp2](#) – веб-сервер, встроенный в [Node.js](#).
- [Go HTTP](#) – веб-сервер, встроенный в [Go](#).

Клиенты:

- веб-браузер, работающий на ПК или переносном устройстве (пр.: кПК)
- программы, самостоятельно обращающиеся к веб-серверам для получения обновлений или другой информации (пр.: антивирус запрашивает обновление БД)
- мобильный телефон, получающий доступ к ресурсам веб-сервера через WAP
- др. цифровые устройства и бытовая техника.

Отличие веб-сервера от сервера приложений: веб-сервер предназначен для обслуживания статических страниц (пр.: HTML, CSS), а сервер приложений отвечает за генерацию динамического содержимого путем выполнения кода на стороне сервера (пр.: JSP, EJB и др.).

Доменное имя – [domain name](#) – символическое имя, служащее для идентификации областей, которые являются единицами административной автономии в сети Интернет, в составе вышестоящей по иерархии такой области. Каждая такая область – домен. Общее пространство имен функционирует благодаря DNS.

Доменные адреса дают возможность адресации Интернет-узлов и расположенным на них ресурсам (веб-сайтам, серверам, серверам эл. почты, другим службам) быть предоставленными в удобной для человека форме.

Структура полного доменного имени:

Непосредственное имя домена + имена всех доменов, в которые он входит, разделенные точками.

FQDN (fully qualified domain name) – полное доменное имя, т.е. не имеющее неоднозначностей в определении, включающее в себя имена всех родительских доменов иерархии DNS.

В DNS, а точнее в zone file, FQDN завершаются точкой, т.е. включают корневое имя «.», которое является безымянным (на практике опускается).

Различия между FQDN и доменным именем проявляются при именовании доменов второго, третьего и т.д. уровней. Для FQDN обязательно указать домены более высокого уровня.

В DNS-записях доменов (для перенаправления (трансляция порт-адрес – технология трансляции порт-адреса в зависимости от TCP-UDP-порта получателя), почтовых серверов и т.д.) всегда используется FQDN.

Доменная зона – совокупность доменных имен определенного уровня, входящих в конкретный домен (термин применяется в тех. сфере при настройке DNS-серверов: поддержание, делегирование и трансфер зоны).

DNS-рэзерверы – отвечают на вопросы, касающиеся любых зон.

Служба whois – позволяет узнать владельца (админа) домена.

Дроп-домен – у него истек срок регистрации, и теперь он свободен.

Виды:

- Тематические домены (gTLD). [Общие домены верхнего уровня](#) (gTLD) управляются организацией [ICANN](#).
- Интернационализированные домены (IDN). Доменные имена, которые содержат символы национальных алфавитов. [IDN](#) верхнего уровня управляются и находятся под контролем [ICANN](#).
- Национальные домены (ccTLD). [Национальные домены верхнего уровня](#) (ccTLD) делегированы соответствующим национальным регистраторам, которые устанавливают правила регистрации в них либо сами, либо согласно указаниям правительства. Управляющей организацией является [IANA](#).
- Зарезервированные доменные имена. Документ [RFC 2606](#) (Reserved Top Level DNS Names — Зарезервированные имена доменов верхнего уровня) определяет названия доменов, которые следует использовать в качестве примеров (например, в документации), а также для тестирования. Кроме [example.com](#), [example.org](#) и [example.net](#), в эту группу также входят [.test](#), [.invalid](#) и др.
- Длинные доменные имена. Размер доменного имени ограничивается по административным и техническим причинам. Обычно разрешается [регистрация доменов](#) длиной до 63 символов. В некоторых странах можно регистрировать домены длиной до 127 знаков.

Apache HTTP-сервер – [апач](#) – свободный веб-сервер.

Это кроссплатформенное ПО, поддерживает операционные системы [Linux](#), [BSD](#), [Mac OS](#), [Microsoft Windows](#), [Novell NetWare](#), [BeOS](#).

Его основные достоинства – надежность и гибкость конфигурации. Он позволяет переключать внешние модули для предоставления данных, использовать СУБД для аутентификации пользователей, модифицировать сообщения об ошибках и т.д. Поддерживает IPv4.

Архитектура:

- Ядро – включает в себя основные функциональные возможности (обработка конфигурационных файлов, протокол HTTP и система загрузки модулей). Язык – С.
- Система конфигурации – основана на текстовых конфигурационных файлах, имеет 3 уровня конфигурации:
 - К. сервера (`httpd.conf`) – директивы к. сгруппированы в 3 осн. раздела: 1) управляющие процессом Apache в целом (глобальное окружение); 2) определяющие параметры «главного» сервера, или сервера по умолчанию, который отвечает на запросы, которые не обрабатываются виртуальными хостами (определяют т.ж. установки по умолчанию для всех остальных виртуальных хостов); 3) установки для виртуальных хостов, позволяющие обрабатывать запросы Web одним единственным сервером Apache, но направлять по разным адресам IP или именам хостов.
 - К. виртуального хоста (`httpd.conf` с версии 2.2, `extra/httpd-vhosts.conf`).
 - К. уровня каталога (`.htaccess`).

Имеет собственный язык конфигурационных файлов, основанный на блоках директив. Практически все параметры ядра могут быть изменены через конфигурационные файлы, вплоть до управления MPM. Большая часть модулей имеет собственные параметры.

Часть модулей использует в своей работе конфигурационные файлы операционной системы (например [/etc/passwd](#) и [/etc/hosts](#)).

Помимо этого, параметры могут быть заданы через ключи [командной строки](#).

- **Многопроцессорные модели (MPM)**.

Для веб-сервера Apache существует множество моделей [симметричной многопроцессорности](#).

Вот основные из них:

Название	Разработчик	Поддерживаемые OS	Описание	Назначение	Статус
worker	Apache Software Foundation	Linux, FreeBSD	Гибридная многопроцессорно-многопоточная модель. Сохраняя стабильность многопроцессорных решений, она позволяет обслуживать большое число клиентов с минимальным использованием ресурсов.	Среднезагруженные веб-серверы.	Стабильный.
pre-fork	Apache Software Foundation	Linux, FreeBSD	MPM, основанная на предварительном создании отдельных процессов, не использующая механизм threads.	Большая безопасность и стабильность за счёт изоляции процессов друг от друга, сохранение совместимости со старыми библиотеками, не поддерживающими threads.	Стабильный.
perchild	Apache Software Foundation	Linux	Гибридная модель, с фиксированным количеством процессов.	Высоконагруженные серверы, возможность запуска дочерних процессов используя другое имя пользователя для повышения безопасности.	В разработке, нестабильный.
netware	Apache Software Foundation	Novell NetWare	Многопоточная модель, оптимизированная для работы в среде NetWare.	Серверы Novell NetWare	Стабильный.
winnt	Apache Software Foundation	Microsoft Windows	Многопоточная модель, созданная для операционной системы Microsoft Windows.	Серверы под управлением Windows Server.	Стабильный.
Apache-ITK	Steinar H. Gunderson	Linux, FreeBSD	MPM, основанная на модели prefork. Позволяет запуск каждого виртуального хоста под отдельными uid и gid.	Хостинговые серверы, серверы, критичные к изоляции пользователей и учёту ресурсов.	Стабильный.
peruser	Sean Gabriel Heacock	Linux, FreeBSD	Модель, созданная на базе MPM perchild. Позволяет запуск каждого виртуального хоста под отдельными uid и gid. Не использует потоки.	Обеспечение повышенной безопасности, работа с библиотеками, не поддерживающими threads.	Стабильная версия от 4 октября 2007 года, экспериментальная — от 10 сентября 2009 года.
event	Apache Software Foundation	Linux, FreeBSD	Модель использует threads и thread-safe polling основана на worker. Предназначен для одновременного обслуживания большего количества запросов путем передачи некоторой обработки в потоки слушателей, освобождая рабочие потоки для обслуживания новых запросов.	Обеспечение повышенной производительности. Не очень хорошо работает на старых платформах, в которых отсутствует хорошая многопоточность, но требование EPoll или KQueue делает это спорным.	Стабильный.

- **Система модулей**.

Apache HTTP Server поддерживает [модульность](#). Модули могут быть как включены в состав сервера в момент [компиляции](#), так и загружены динамически, через директивы конфигурационного файла.

В модулях реализуются такие вещи, как:

- Поддержка [языков программирования](#).
- Добавление функций.
- Исправление ошибок или модификация основных функций.
- Усиление [безопасности](#).

Часть веб-приложений, например панели управления [ISPmanager](#) и [VDSmanager](#) реализованы в виде модуля Apache.

- **Механизм виртуальных хостов**.

Apache имеет встроенный механизм виртуальных [хостов](#). Он позволяет полноценно обслуживать на одном [IP-адресе](#) множество [сайтов \(доменных имён\)](#), отображая для каждого из них собственное содержимое.

Для каждого виртуального хоста можно указать собственные настройки ядра и модулей, ограничить доступ ко всему сайту или отдельным файлам. Некоторые МРМ, например Apache-ITK, позволяют запускать [процесс](#) httpd для каждого виртуального хоста с отдельными идентификаторами [uid](#) и [guid](#).

Также существуют модули, позволяющие учитывать и ограничивать ресурсы [сервера](#) ([CPU](#), [RAM](#), [трафик](#)) для каждого виртуального хоста.

Функциональные возможности:

- [Интеграция с другим ПО и языками программирования](#)

Существует множество модулей, добавляющих к Apache поддержку различных [языков программирования](#) и систем разработки.

К ним относятся:

- [PHP](#) (mod_php).
- [Python](#) ([mod python](#), [mod wsgi](#)).
- [Ruby](#) (apache-ruby).
- [Perl](#) ([mod perl](#)).
- [ASP](#) (apache-asp).
- [Tcl](#) (rivet)

Кроме того, Apache поддерживает механизмы [CGI](#) и [FastCGI](#), что позволяет исполнять программы на практических всех языках программирования, в том числе [C](#), [C++](#), [Lua](#), [sh](#), [Java](#).

- [Безопасность](#)

Apache имеет различные механизмы обеспечения безопасности и разграничения доступа к данным. Основными являются:

- Ограничение доступа к определённым каталогам или файлам.
- Механизм [авторизации](#) пользователей для доступа к каталогу на основе HTTP-аутентификации ([mod_auth_basic](#)) и [digest-аутентификации](#) ([mod_auth_digest](#)).
- Ограничение доступа к определённым каталогам или всему серверу, основанное на [IP-адресах](#) пользователей.
- Запрет доступа к определённым типам файлов для всех или части пользователей, например запрет доступа к конфигурационным файлам и файлам баз данных.
- Существуют модули, реализующие авторизацию через [СУБД](#) или [PAM](#).

В некоторых МРМ-модулях присутствует возможность запуска каждого процесса Apache, используя различные [uid](#) и [gid](#) с соответствующими этим пользователям и группам пользователям.

Также существует механизм [suexec](#), используемый для запуска [скриптов](#) и [CGI](#)-приложений с правами и идентификационными данными пользователя.

Для реализации [шифрования](#) данных, передающихся между клиентом и сервером, используется механизм [SSL](#), реализованный через библиотеку [OpenSSL](#). Для удостоверения подлинности веб-сервера используются сертификаты [X.509](#).

Существуют внешние средства обеспечения безопасности, например [mod_security](#).

- [Интернационализация](#)

Начиная с версии 2.0 появилась возможность определения сервером [локали](#) пользователя. Сообщения об ошибках и событиях, посылаемые браузеру, теперь представлены на нескольких языках и используют [SSI](#)-технологию.

Также, можно реализовать средствами сервера отображение различных страниц для пользователей с различными локалами. Apache поддерживает множество кодировок, в том числе [Unicode](#), что позволяет использовать страницы, созданные в любых кодировках и на любых языках.

- Обработка событий

Администратор может установить собственные страницы и обработчики для всех [HTTP](#)-ошибок и событий, таких как 404 (Not Found) или 403 (Forbidden). В том числе существует возможность запуска [скриптов](#) и отображения сообщений на разных языках.

[SSI](#) - В версиях 1.3 и старше был реализован механизм Server Side Includes, позволяющий динамически формировать [HTML](#)-документы на стороне сервера. Управлением SSI занимается модуль [mod_include](#), включённый в базовую поставку Apache.

База данных – [Data Base](#) – совокупность данных, хранимых в соответствии со схемой данных, манипулирование с которыми осуществляют в соответствии с правилами средств модулирования данных.

База данных — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

Признаки:

- БД хранится и обрабатывается в вычислительной системе
- Данные БД логически структурированы для облегчения работы с ними (систематизация: выделение составных частей, построение связей между ними, типизация, семантика и допустимые операции)
- БД включает схему, или метаданные, описывающие ее логическую структуру в формальном виде (в соответствии с некоторой метамоделью)

Виды БД:

По модели данных:

- [иерархические](#)
- [объектные](#) или [объектно-ориентированные](#)
- [объектно-реляционные](#)
- [реляционные](#)
- [сетевые](#)
- функциональные

По системе хранения:

- традиционные – conventional DB – хранят данные во вторичной памяти
- [резидентные](#) – все данные на стадии исполнения находятся в оперативной памяти
- третичные – tertiary DB – хранят данные на отсоединяемых устройствах массового хранения

Для некоторых БД строятся [специализированные СУБД, либо доп. возможности СУБД](#):

- пространственные (spatial DB) – базы с пространственными свойствами сущности предметной области
- временные – поддерживают к-л аспект времени (не считая времени, определяемого пользователем)

По степени распределенности:

- централизованные (сосредоточенные) – полностью поддерживаемые на одном оборудовании

- распределенные (distributed DB) – на разных узлах комп. сети по к-л критерию. Среди них выделяют:
 - сегментированные – разделенные на части под управлением различных экземпляров СУБД по к-л критерию
 - тиражированные (реплицированные) – в них одни и те же данные разнесены под управление разных экземпляров СУБД
 - неоднородные (heterogeneous distributed DB) – фрагменты распределенной базы в разных узлах сети поддерживаются средствами более одной СУБД

По способам организации хранения:

- циклические – записывают новые данные заместо устаревших
- потоковые

SOLID

Хороший код:

- Масштабируемый, легко вносить изменения
- Порог вхождения в проект – новому человеку легко разобраться
- Простой, без усложнений
- **S – Single Responsibility Principle** – один объект – одна задача, одна зона ответственности

Декомпозиция на модули. Разделяем модель данных и поведение.

Иначе – путаница, трудно вносить изменения. Декомпозированный код легче читать, править, тестировать, мерджить.

God-object – делает много задач.

- **O – Open/Closed Principle** – программные сущности открыты для расширения и закрыты для изменения.

Новый функционал вводим за счет создания новых сущностей путем наследования, композиции, а не изменением старых.

Если и изменяем код, необходимо регрессионное тестирование.

- **L – Liskov's Substitution Principle** - функции, сущности, которые используют родительский тип, должны так же работать и с дочерними классами, при этом не должна нарушаться логика программы. Наследуемый класс должен дополнять, а не замещать поведение базового класса.
- **I – Interface Segregation Principle** – программные сущности не должны зависеть от методов, которые они не используют.

Код менее связанный, сущности не зависят от методов, которые к ним не относятся.

- **D – Dependency Inversion Principle** – модули высокого уровня не должны зависеть от модулей нижнего уровня, все они должны зависеть от абстракций, а абстракции не должны зависеть от деталей, наоборот – детали должны зависеть от абстракций.

Парадигма ООП

Процедурный подход – программе даются данные, она выполняет какие-то функции и возвращает какой-то результат.

В ООП (объектно-ориентированном программировании) есть класс, в нем есть объекты, у которых есть свойства. Объект – конкретный представитель класса со своими обозначенными свойствами. Объект может совершать некоторые действия – методы.

Объекты здесь не только классы (и объекты), но и сущности более высокого уровня – модули, пакеты, неймспейсы, сабсистемы, система, сабпакеты и т.д.

Основные концепции ООП (принципы):

1. Инкапсуляция

Сам класс – капсула, которая содержит свойства и методы для работы с этими свойствами.

В объект (класс) объединяются методы и данные.

Скрытие – private (можно использовать только внутри класса, использовать извне нельзя) и public (модификаторы доступа) (одна из трактовок инкапсуляции).

Итого, это и объединение методов и данных в один объект, и скрытие этих методов и данных от внешнего воздействия. Объект не должен изменяться ничем извне, кроме методов самого объекта (внутри).

Get, set – создаются, чтобы получать доступ к приватным свойствам (получать и изменять).

Пример: имя пользователя и пароль можно получать и менять, а ID – только получать.

Из инкапсуляции вытекает множество паттернов GRASP и GoF.

Lowcoplin

2. Наследование

Новые классы наследуют черты от родительского(-их) класса(-ов) и имеют собственные свойства.

Не можем унаследоваться от нескольких классов (в большинстве языков).

Конструктор т.ж. по умолчанию наследуется.

3. Полиморфизм

В общем, это способность функции работать с данными разных типов.

Есть два типа:

- Полиметрический (истинный)

Когда одна и та же функция с одним и тем же телом способна принимать в качестве параметра данные разных классов.

Пример: у нас есть несколько классов: родительский класс «человек», от него наследуется класс «рабочник», а уже от него – класс «программист». Всем классам нужно написать функцию приветствия, возвращающую «Привет» + «я» + название класса + имя объекта.

Принимать данные от каждого массива – плохая идея, поэтому мы применяем полиморфизм, чтобы каждый объект, унаследованный от класса «человек» смог поздороваться.

Создаем массив со всеми объектами, создаем функцию массового приветствия с аргументом массивом.

- ad-hoc (мнимый) – класс имеет несколько методов, например, работающих с разными типами данных, а так же класс, в котором мы делаем преобразование дочернего класса до родительского

Это приведение данных к тому типу, с которым работаем метод; перегрузка метода – когда он существует с одинаковыми названиями, но разными параметрами.

Есть высказывание, что все if можно заменить на полиморфизм. Он помогает в разы уменьшать размер программы.

Рефакторинг, coding by exception.

Еще сейчас выделяют 4 принцип – абстракция.

Помимо наследования существуют еще два способа взаимодействия классов:

- Композиция

Пример: есть класс автомобиль, внутри которого есть объект класса двигатель, а также массив объектов класса колесо. Двигатель и колеса не могут существовать отдельно от автомобиля.

- Агрегация

Все тот же автомобиль. Однако добавляется класс «елочка-освежитель». Он может существовать отдельно от класса «автомобиль». Тогда елочка будет отдельным свойством у класса «автомобиль», обращающимся к собственному классу – «освежитель». А в конструкторе объекта мы к нему обращаемся через свойство.

Абстрактные классы и интерфейсы

Интерфейс – как оглавление в учебнике. Он говорит, что нужно сделать, но не говорит как. Из него нельзя создать объект.

Абстрактные классы похожи на интерфейс, кроме того, в них можно определять абстрактные методы (аналоги методов в интерфейсе, т.е. без реализации), но в них можно создавать и обычные методы с реализацией и логикой. Класс, образованный от абстрактного, наследует все обычные методы, а т.ж. абстрактные.

generic – обобщение <ожидаемый тип данных>

Пример для закрепления: Dependency Injection (внедрение зависимостей) (паттерн)

У нас есть два слоя приложения:

1) работа с БД (получить, записать, изменить, удалить из репозитория). Реализации – через MongoRepository и MySqlRepository.

2) БЛ. Здесь вопрос: какой репозиторий выбрать? Лучше, чтобы сервисный слой не знал, с каким репозиторием работает. Т.е. мы делаем имплементацию БД извне, пр. на уровне конфигурации. Так нам не надо менять сервисный слой, достаточно поправить конфигурацию.

interface UserRepo

 getUsers

class UserMongoDBRepo implements UserRepo

class UserService

 userRepo: UserRepo

 constructor (userRepo: UserRepo):

 this.userRepo = userRepo

 filterUserByAge(age: number)

```

const users = this.UserRepo.getUsers
    // какая-то логика по фильтрации
    console.log(users)
const userService = new UserService (new UserMongoDBRepo)
userService.filterUserByAge (age=13)

```

Мы можем передать в конструктор аргумент new UserMySQLDBRepo, то есть извне, а сам сервис не трогали.

Еще пример с паттерном Singleton (нужно, чтоб было только одно подключение к БД):

```

class Database

url: string

private static instance: Database

constructor(url: string)

if Database.instance

    return Database.instance

this.url = Math.random()

Database.instance = this

const db1 = new Database()

const db2 = new Database()

console.log(db1.url)

console.log(db2.url)

```

Выведет два одинаковых рандомных числа, т.е. вход один.

Паттерны проектирования

П. пр. – это часто встречающееся решение (концепция) определенной проблемы при проектировании архитектуры программ.

Польза:

- проверенные решения (не надо изобретать велосипед)
- стандартизация кода (проще для понимания, меньше просчетов, а значит, и костылей)
- общий программистский словарь (удобство понимания)

Критика:

- костыль для языка с недостаточным уровнем абстракции («слабого» в данной ситуации)
- могут быть неэффективны, если применяются без адаптации под конкретный проект
- новички начинают «пишать» паттерны везде, даже если можно сделать проще

Классификация:

- Самые простые и низкоуровневые – идиомы – реализуемы в рамках одного языка.

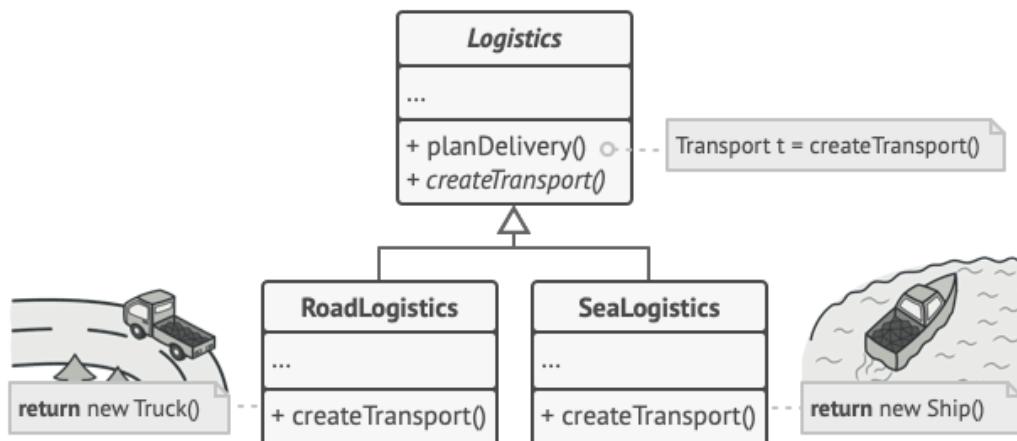
- Самые высокоуровневые – архитектурные паттерны – применимы для всех языков и служат для проектирования программ в целом, не отдельных элементов.
- Классификация по предназначению:
 - Порождающие – служат для быстрого создания объектов без внесения в программу лишних зависимостей
 - Структурные – показывают различные способы построения связей между объектами
 - Поведенческие – заботятся об эффективной коммуникации между объектами

Каталог паттернов:

- **Порождающие**
 - **Фабричный метод (Factory method)** – он же «виртуальный конструктор» - порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Решаемая проблема: если код задействует один класс (пр.: грузовик), а потом надо внедрить новый класс (пр.: пароход), то придется переписывать весь код, создавать условные операторы, которые работают в зависимости от класса.

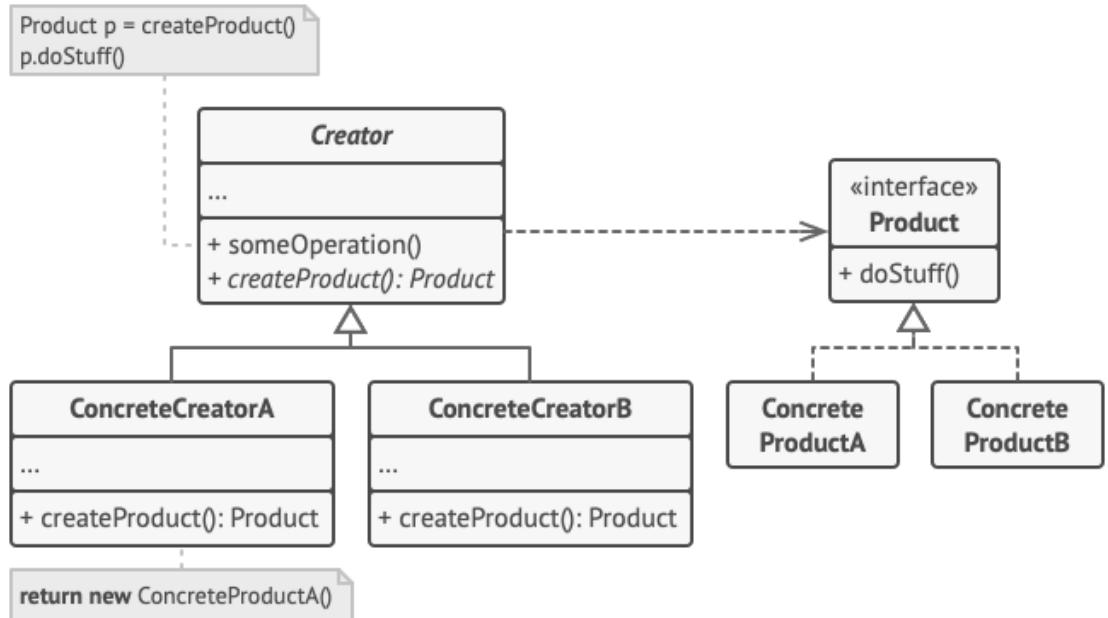
Решение: объекты создаются не напрямую, а через вызов фабричного метода (при этом фабричный метод использует все те же операторы, например, new).



Подклассы смогут производить объекты различных классов, следующих одному и тому же интерфейсу.

Например, классы Грузовик и Судно реализуют интерфейс Транспорт с методом доставить. Каждый из этих классов реализует метод по-своему: грузовики везут грузы по земле, а суда — по морю. Фабричный метод в классе «Дорожной Логистики» вернёт объект-грузовик, а класс «Морской Логистики» — объект-судно.

Для клиента фабричного метода нет разницы между этими объектами, так как он будет трактовать их как некий абстрактный Транспорт. Для него будет важно, чтобы объект имел метод доставить, а как конкретно он работает — не важно.

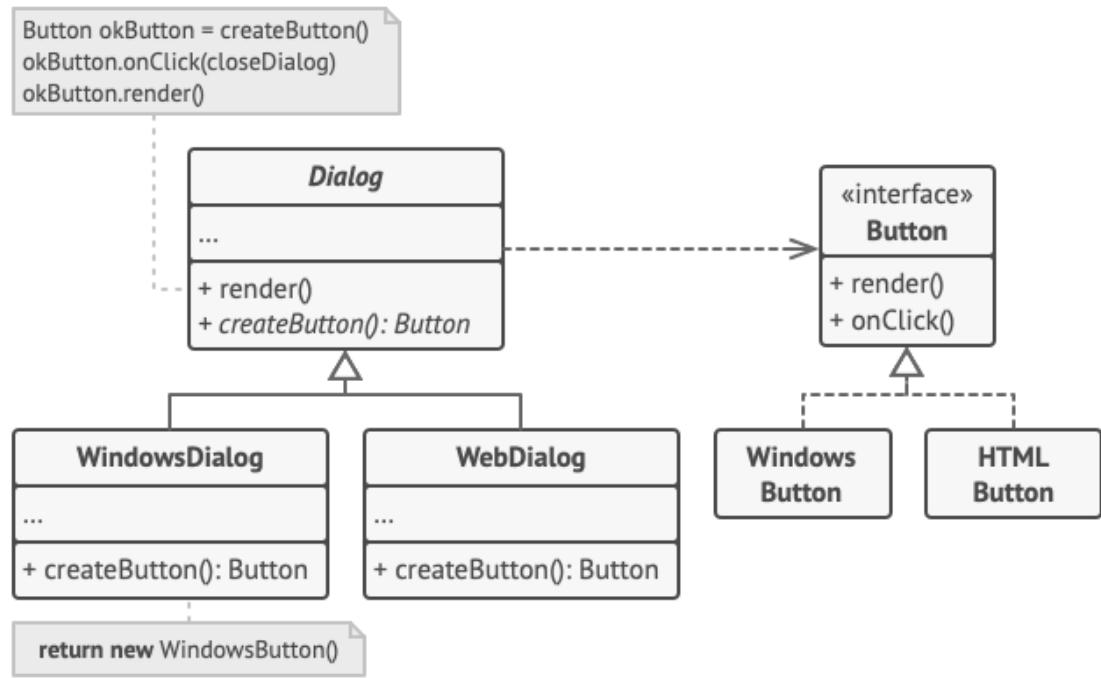


1. Продукт определяет общий интерфейс объектов, которые может произвести создатель и его подклассы.
2. Конкретные продукты содержат код различных продуктов, которые отличаются реализацией, но имеют общий интерфейс.
3. Создатель объявляет фабричный метод, который должен возвращать новые объекты продуктов. Важно, чтобы тип результата совпадал с общим интерфейсом продуктов. Обычно, ФМ – абстрактный, но может создавать и типовой продукт. Создатель, помимо создания продуктов, может выполнять и сторонние функции.
4. Конкретные создатели по-своему реализуют фабричный метод, производя те или иные конкретные продукты.

Фабричный метод можно и переписать так, чтобы возвращать существующие объекты из какого-то хранилища или кэша.

Пример:

В этом примере Фабричный метод помогает создавать кросс-платформенные элементы интерфейса, не привязывая основной код программы к конкретным классам элементов.



Фабричный метод объявлен в классе диалогов. Его подклассы относятся к различным операционным системам. Благодаря фабричному методу, вам не нужно переписывать логику диалогов под каждую систему. Подклассы могут наследовать почти весь код из базового диалога, изменяя типы кнопок и других элементов, из которых базовый код строит окна графического пользовательского интерфейса.

Базовый класс диалогов работает с кнопками через их общий программный интерфейс. Поэтому, какую вариацию кнопок ни вернул бы фабричный метод, диалог останется рабочим. Базовый класс не зависит от конкретных классов кнопок, оставляя подклассам решение о том, какой тип кнопок создавать.

Стоит применять:

- когда заранее не известны типы и зависимости объектов, с которыми должен работать код (код производства продуктов идет отдельно от остального кода, так что его можно расширять, не трогая основной код; например, чтобы добавить поддержку нового продукта, вам нужно создать новый подкласс и определить в нем фабричный метод, возвращая оттуда экземпляр нового продукта)
- когда вы хотите дать возможность пользователям расширять части вашего фреймворка или библиотеки (через наследование, создание и использование подклассов)
- когда вы хотите экономить системные ресурсы, повторно используя уже созданные объекты, вместо создания новых (т.к. в таком случае нужен метод, который будет и отдавать существующие объекты, и новые)

Шаги реализации:

1. Приведите все создаваемые продукты к общему интерфейсу.
2. В классе, который производит продукты, создайте пустой фабричный метод. В качестве возвращаемого типа укажите общий интерфейс продукта.

3. Затем пройдитесь по коду класса и найдите все участки, создающие продукты. Поочерёдно замените эти участки вызовами фабричного метода, перенося в него код создания различных продуктов.

4. В фабричный метод, возможно, придётся добавить несколько параметров, контролирующих, какой из продуктов нужно создать.

На этом этапе фабричный метод, скорее всего, будет выглядеть удручающе. В нём будет жить большой условный оператор, выбирающий класс создаваемого продукта. Но не волнуйтесь, мы вот-вот исправим это.

5. Для каждого типа продуктов заведите подкласс и переопределите в нём фабричный метод. Переместите туда код создания соответствующего продукта из суперкласса.

6. Если создаваемых продуктов слишком много для существующих подклассов создателя, вы можете подумать о введении параметров в фабричный метод, которые позволят возвращать различные продукты в пределах одного подкласса.

Например, у вас есть класс Почта с подклассами АвиаПочта и НаземнаяПочта, а также классы продуктов Самолёт, Грузовик и Поезд. Авиа соответствует Самолётам, но для НаземнойПочты есть сразу два продукта. Вы могли бы создать новый подкласс почты для поездов, но проблему можно решить и по-другому. Клиентский код может передавать в фабричный метод НаземнойПочты аргумент, контролирующий тип создаваемого продукта.

7. Если после всех перемещений фабричный метод стал пустым, можете сделать его абстрактным. Если в нём что-то осталось — не беда, это будет его реализацией по умолчанию.

Плюсы и минусы:

- + Избавляет класс от привязки к конкретным классам продуктов.
- + Выделяет код производства продуктов в одно место, упрощая поддержку кода.
- + Упрощает добавление новых продуктов в программу.
- + Реализует принцип открытости/закрытости.
- Может привести к созданию больших параллельных иерархий классов, так как для каждого класса продукта надо создать свой подкласс создателя.

Отношения с другими паттернами:

- Многие архитектуры начинаются с применения Фабричного метода (более простого и расширяемого через подклассы) и эволюционируют в сторону Абстрактной фабрики, Прототипа или Строителя (более гибких, но и более сложных).
- Классы Абстрактной фабрики чаще всего реализуются с помощью Фабричного метода, хотя они могут быть построены и на основе Прототипа.
- Фабричный метод можно использовать вместе с Итератором, чтобы подклассы коллекций могли создавать подходящие им итераторы.
- Прототип не опирается на наследование, но ему нужна сложная операция инициализации. Фабричный метод, наоборот, построен на наследовании, но не требует сложной инициализации.
- Фабричный метод можно рассматривать как частный случай Шаблонного метода. Кроме того, Фабричный метод нередко бывает частью большого класса с Шаблонными методами.

Примеры реализации – см. в [источнике](#).

```
from __future__ import annotations
from abc import ABC, abstractmethod

class Creator(ABC):
    """
        Класс Создатель объявляет фабричный метод, который должен
        возвращать объект
        класса Продукт. Подклассы Создателя обычно предоставляют
        реализацию этого
        метода.
    """

    @abstractmethod
    def factory_method(self):
        """
            Обратите внимание, что Создатель может также обеспечить
            реализацию
            фабричного метода по умолчанию.
        """

        pass

    def some_operation(self) -> str:
        """
            Также заметьте, что, несмотря на название, основная обязанность
            Создателя не заключается в создании продуктов. Обычно он
            содержит
            некоторую базовую бизнес-логику, которая основана на объектах
            Продуктов,
            возвращаемых фабричным методом. Подклассы могут косвенно
            изменять эту
            бизнес-логику, переопределяя фабричный метод и возвращая из
            него другой
            тип продукта.
        """

        # Вызываем фабричный метод, чтобы получить объект-продукт.
        product = self.factory_method()

        # Далее, работаем с этим продуктом.
        result = f"Creator: The same creator's code has just worked with {product.operation()}""

        return result

    """

    Конкретные Создатели переопределяют фабричный метод для того,
    чтобы изменить тип
    результирующего продукта.
    """

class ConcreteCreator1(Creator):
    """
        Обратите внимание, что сигнатура метода по-прежнему использует
        тип
        абстрактного продукта, хотя фактически из метода возвращается
        конкретный
        продукт. Таким образом, Создатель может оставаться независимым от
        конкретных
        классов продуктов.
    """

    def factory_method(self) -> Product:
        return ConcreteProduct1()

class ConcreteCreator2(Creator):
    def factory_method(self) -> Product:
        return ConcreteProduct2()
```

```

class Product(ABC):
    """
    Интерфейс Продукта объявляет операции, которые должны выполнять
    все
    конкретные продукты.
    """

    @abstractmethod
    def operation(self) -> str:
        pass


    """
    Конкретные Продукты предоставляют различные реализации интерфейса
    Продукта.
    """

class ConcreteProduct1(Product):
    def operation(self) -> str:
        return "[Result of the ConcreteProduct1]"

class ConcreteProduct2(Product):
    def operation(self) -> str:
        return "[Result of the ConcreteProduct2]"

def client_code(creator: Creator) -> None:
    """
    Клиентский код работает с экземпляром конкретного создателя,
    хотя и через
    его базовый интерфейс. Пока клиент продолжает работать с
    создателем через
    базовый интерфейс, вы можете передать ему любой подкласс
    создателя.
    """

    print(f"Client: I'm not aware of the creator's class, but it still works.\n"
          f"[{creator.some_operation()}", end="")

if __name__ == "__main__":
    print("App: Launched with the ConcreteCreator1.")
    client_code(ConcreteCreator1())
    print("\n")

    print("App: Launched with the ConcreteCreator2.")
    client_code(ConcreteCreator2())

```

Output.txt: Результат выполнения

```

App: Launched with the ConcreteCreator1.
Client: I'm not aware of the creator's class, but it still works.
Creator: The same creator's code has just worked with [Result of the ConcreteProduct1]

App: Launched with the ConcreteCreator2.
Client: I'm not aware of the creator's class, but it still works.
Creator: The same creator's code has just worked with [Result of the ConcreteProduct2]

```

- **Абстрактная фабрика (Abstract factory)**

Это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

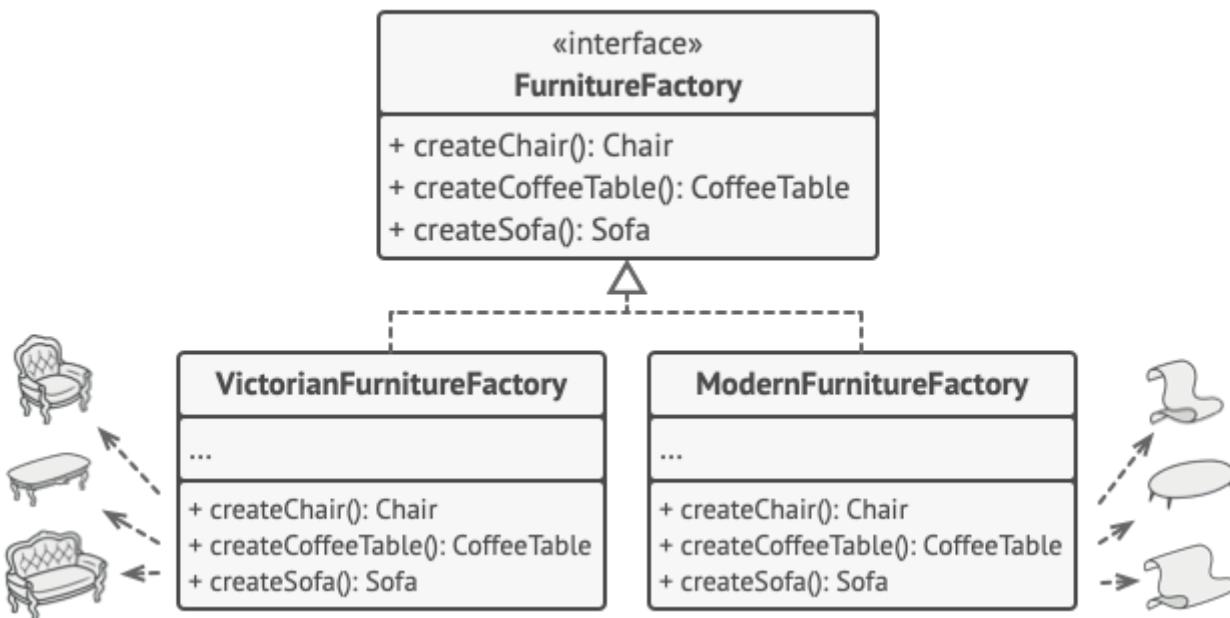
Решаемая проблема: пример – мебельный магазин. Есть:

- семейство зависимых продуктов (кресло+диван+столик)
- несколько вариаций этого семейства (стили модерн, классика и т.д.)

Нужен способ создавать объекты продуктов, чтобы они сочетались с другими продуктами того же семейства, не внося изменений в существующий код при создании новых продуктов или семейств.

Решение: выделяем общие интерфейсы для отдельных продуктов, составляющих семейства (у кресел свой одинаковый интерфейс и т.д.).

Создаем абстрактную фабрику – общий интерфейс, который содержит методы создания всех продуктов семейства. Эти операции должны возвращать абстрактные типы продуктов, представленные интерфейсами (кресла, диваны и т.д.)



Для каждой вариации продуктов мы должны создать свою собственную фабрику, реализовав абстрактный интерфейс. Фабрики создают продукты одной вариации (ФабрикаМодерн – КреслаМодерн, ДиванМодерн и т.д.).

Клиентский код должен работать как с фабриками, так и с продуктами только через их общие интерфейсы, что позволяет подавать в классы любой тип фабрики и производить любые продукты.

Кто же создает объекты конкретных фабрик, если клиентский код создает только интерфейсы самих фабрик? Обычно программа создает конкретный объект фабрики при запуске, причем тип фабрики выбирается, исходя из параметров окружения или конфигурации.

- Строитель (Builder)
- Прототип (Prototype)
- Одиночка (Singleton)
- Структурные
 - Адаптер (Adapter)
 - Мост (Bridge)
 - Компоновщик (Composite)
 - Декоратор (Decorator)
 - Фасад (Facade)
 - Легковес (Flyweight)
 - Заместитель (Proxy)
- Поведенческие
 - Цепочка обязанностей (Chain of Responsibility)
 - Команда (Command)
 - Итератор (Iterator)

- Посредник (Mediator)
- Снимок (Memento)
- Наблюдатель (Observer)
- Состояние (State)
- Стратегия (Strategy)
- Шаблонный метод (Template Method)
- Посетитель (Visitor)

JSON

JavaScript Object Notation – текстовый формат обмена данными, основанный на JavaScript. Используется в REST API. Альтернатива – XML.

Значения: числа, строки, массивы, JSON-объекты, литералы (лог. зн. true, false, null).

JSON-объект – неупорядоченное множество пар {"ключ" : значение}, взаимодействие с которыми происходит, как со словарями.

Ключ – название параметра, который мы передаем серверу. Он служит маркером для принимающей стороны запрос системы, чтобы она поняла, что мы ей отправили.

Ключ всегда строка. Строки в кавычках. Порядок – любой.

Обратиться к ключу можно через:

- get() – если ключа нет, то пустое значение (NoneType), также имеет второй передаваемый аргумент (возвращается, если ключа нет)
- dict[key] – если ключа нет, то KeyError

JSON-массив: [“MALE”, “FEMALE”].

Нет ключей, набор значений, обращение по номеру элемента. Нельзя менять местами данные – упорядоченное множество.

Well Formed JSON – синтаксически правильный.

Правила:

1. Данные написаны в виде пар «ключ: значение»
2. Данные разделены запятыми
3. Объект находится внутри {фигурных скобок}
4. Массив – внутри [квадратных]

Для проверки синтаксиса - JSON Validator, JSON Formatter (он еще и форматирует).

Бизнес-логика

Это правила работы программы, взаимодействия ее внутренних составляющих, а также взаимодействие со внешними элементами.

----- не по теме, но интересно -----

- [] How does the Internet work?
- [] How does a computer work?
- [] What is “back-end”?
- [] Hosting
- [] API
- [] Server
- [] DNS

- [] HTTP
- [] Browser
- [] Domain name
- [] About Python
- [] IDE (e.g. PyCharm, VS Code, Visual Studio, IntelliJ Idea)
- [] Terminal (PowerShell, Bash, Zsh)
- [] Framework (Django)
- [] Data Bases (PostgreSQL, MySQL, MS SQL, Azure Cloud, SQL; MongoDB, Redis, Cassandra, AWS, Firebase)
- [] Data Structures
- [] Authentication (OAuth 2.0, JWT; providers - Auth0, Firebase)
- [] Patterns of Programming
- [] Testing, QA (frameworks)
- [] Package Managers (NuGet, npm, yarn)
- [] Version Control Systems (git, GitHub, TFS, BitBucket)
- [] Web-services
- [] Network Protocols
- [] UI-frameworks
- [] OOP