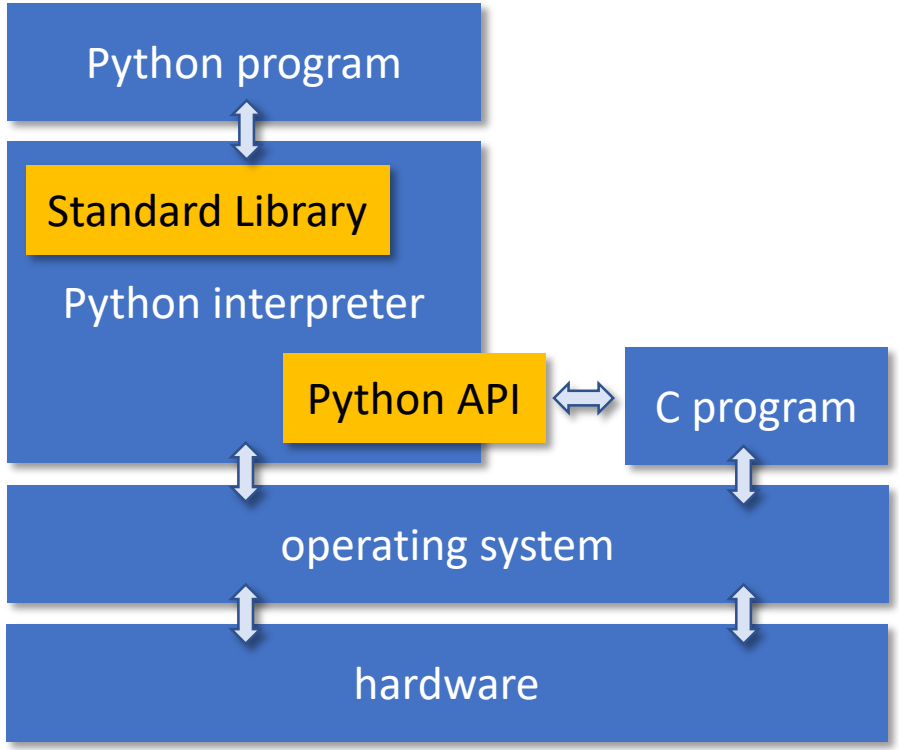
# Python for Beginners

Language Basics

# Python Architecture

- The python program is executed by an interpreter
- The interpreter abstracts the operating system and hardware
- The program can use an extensive standard library
- Python can be extended by C programs via an API



# C# versus Python

- Use { } to indicate blocks
- Variables
  - Must always be declared with data type specified
  - A variable can store only one data type
- Errors can be found early

- Use indentation for blocks
- Variables
  - Can be used instantly
  - A variable can store any data type which can be changed during program execution
- Flexible and short code

# Comments

```
# the hash symbol is used to start a comment
# every comment must start with a hash


""" There are no multiline comments but so called docstrings
These docstrings
 - start with 3 " and
 - must end with 3 " again
docstrings may contain as many line breaks as you want
"""
```

- Use simple comments to describe the code for better understanding
- Use docstrings for API documentations only
  (describing functions and their parameters, return values etc.)

https://www.python.org/dev/peps/pep-0008/#comments

# Statements

```
# a statement ends with the end of the line
print("Hello World")

# you can place one statement after the other with ;
# but its not nice and not common
print("Hello World"); print("Hello again!")

# pycharm would suggest and help you to beautify the code ;-)
print("Hello World")
print("Hello again!")
```

# Variables

```
# variables start living by getting a value
# variables don't have a data type
height = 3 # use spaces before and after operators!
width = 4

# but the values of variables do have a data type!!
# input always returns a string
height = input("Please enter the height: ")
width = input("Please enter the width: ")

# strings cannot be used for multiplications
area = height * width
```

## Terminal Output:

```
Please enter the height: 3
Please enter the width: 4
Traceback (most recent call last):
  File "C:/Users/harald/PycharmProjects/PythonBasics/variables.py", line 11, in <module>
    area = height * width
TypeError: can't multiply sequence by non-int of type 'str'
```

# Type Conversion

```
# convert the string to an int
height = input("Please enter the height: ")
height = int(height)
# short way
width = int(input("Please enter the width: "))

area = height * width

# convert the int result to a string again
print("area = " + str(area))

# get the data type of a variable
print(type(area))
```

Standard data types:

int
float
str
bool
bytes
complex

More about standard types:
https://docs.python.org/3.6/library/stdtypes.html

# Type Conversion and Math Functions

| | |
|---|---|
| ascii() | Returns a string containing a printable representation of an object |
| bin() | Converts an integer to a binary string |
| bool() | Converts an argument to a Boolean value |
| chr() | Returns string representation of character given by integer argument |
| complex() | Returns a complex number constructed from arguments |
| float() | Returns a floating-point object constructed from a number or string |
| hex() | Converts an integer to a hexadecimal string |
| int() | Returns an integer object constructed from a number or string |
| oct() | Converts an integer to an octal string |
| ord() | Returns integer representation of a character |
| repr() | Returns a string containing a printable representation of an object |
| str() | Returns a string version of an object |
| type() | Returns the type of an object or creates a new type object |

| | |
|---|---|
| abs() | Returns absolute value of a number |
| divmod() | Returns quotient and remainder of integer division |
| pow() | Raises a number to a power |
| round() | Rounds a floating-point value |

# Operators

Same arithmetic operators as in C# but:

| Operator | Example | Meaning | Result |
|---|---|---|---|
| / | a / b | Division | Quotient when a is divided by b.<br>The result always has type float. |
| // | a // b | Floor Division<br>(also called Integer Division) | Quotient when a is divided by b, rounded to the next smallest whole number |
| ** | a ** b | Exponentiation | a raised to the power of b |

Same logical operators as in C# but:

| Operator | Example | Meaning |
|---|---|---|
| not | not x | True if x is False, False if x is True (Logically reverses the sense of x) |
| or | x or y | True if either x or y is True, False otherwise |
| and | x and y | True if both x and y are True, False otherwise |

# True or not true – that’s the question

- Any value that is numerically zero (0, 0.0, 0.0+0.0j) is false
- A non-zero value is true

- An empty string is false
- An non-empty string is true

- An empty list is false
- An non-empty list is true

- The special value denoted by the Python keyword None is false

# Identity Operators

```
x = 1001
y = 1000 + 1

# print the object ids
print(id(x))→ 123162208
print(id(y))→ 131598144

# compare the object values
print(x == y)→ True

# compare the object identities
print (x is y)→ False
```

# Strings and chars

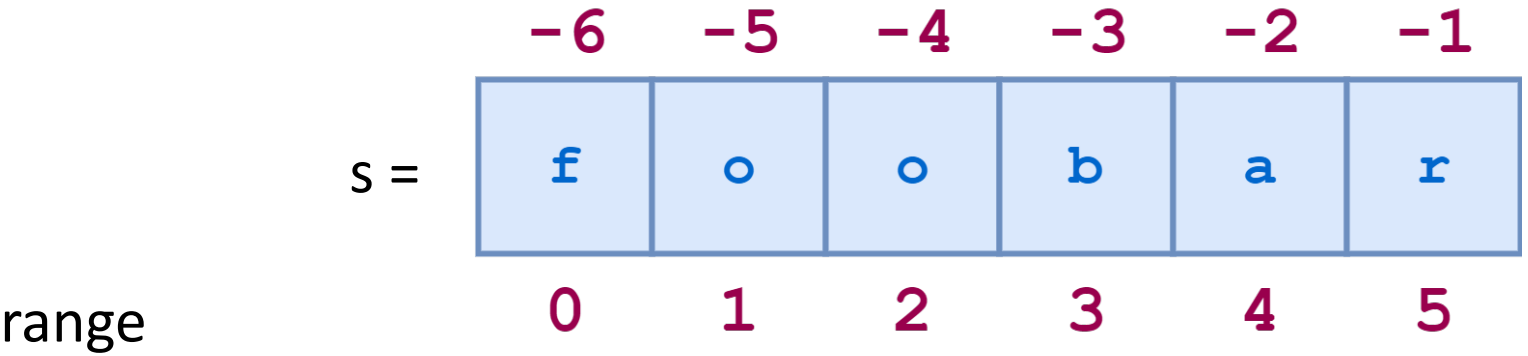
```
# you can use single quotes or double quotes
s = 'HTL'
t = "-"
u = 'Villach'
print(s + t + u) → HTL-Villach

bla = 'bla'
chatterbox = bla * 4
print(chatterbox) → blablablabla

sentence = 'That\'s good for the brain' # escaped quote within a string
print('goo' in sentence) → True

print('z' not in 'xyz') → False
```

# String Indexing, Slicing and Strides




`len(s)` → 6

`s[len(s) - 1]` → r

`s[6]` → string index out of range

`s[-1]` → 'r'

`s[2:5]` → 'oba'

`s[:4]` same as `s[0:4]` → 'foob'

`s[2:]` → 'obar'

`s[0:6:2]` → 'foa' # go from 0 to 6 and skip every second character

# Modifing Strings

- Strings are immutable. Strings cannot be modified!

`s[3] = 'x'` → TypeError: 'str' object does not support item assignment

→Always create a new string:

```
s = s[:3] + 'x' + s[4:]
s = s.replace('b', 'x')
```

- There are many other built-in string methods
- https://docs.python.org/3/library/stdtypes.html#string-methods

# Lists

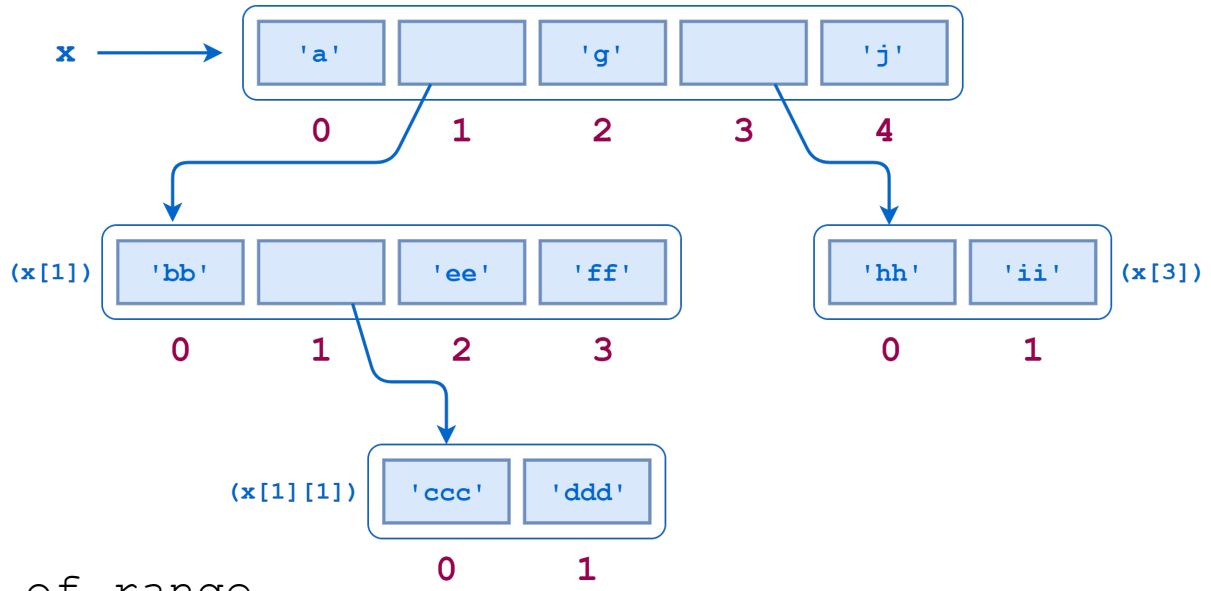
```
empty_list = []
alphabet_list = ['alpha', 'bravo', 'charlie']
mixed_list = ['abc', 123, 21.5, False]

# append to the end
empty_list.append('first entry')
# remove from the list
mixed_list.remove(123)
# clear the list
empty_list.clear()
```

## Lists are mutable:

```
mixed_list[0] = 34
mixed_list[1] = empty_list
Mixed_list[10] = 23
IndexError: list assignment index out of range
```

## Lists can be nested:



# Lists (2)

```
my_list = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
print(my_list[1:4])
```

→`['bar', 'baz', 'qux']`

```
my_list[1:4] = [1.1, 2.2, 3.3, 4.4, 5.5]
print(my_list)
```

→`['foo', 1.1, 2.2, 3.3, 4.4, 5.5, 'quux', 'corge']`

```
my_list[1:6] = ['Bark!']
print(my_list)
```

→`['foo', 'Bark!', 'quux', 'corge']`

# Splitting Strings and Joining Lists

```
abc_str = 'a,b,c'
# split the string in a list of parts
list_of_parts = abc_str.split(',')
print(list_of_parts)
```

→ ['a', 'b', 'c']

```
# join the parts of the list with '-'
abc_str = '-'.join(list_of_parts)
print(abc_str)
```

→ a-b-c

# Dictionaries

```
# an empty dictionary
teachers = {}

# fill with key/value pairs
teachers['woh'] = 'Harald Wolf'
teachers['huo'] = 'Robert Hufsky'

# print the value of key 'huo'
print(teachers['huo'])

keys = teachers.keys()
values = teachers.values()
count = len(teachers)
```

# Dictionaries (2)

```
# another dictionary
person = {}

person['fname'] = 'Joe'
person['lname'] = 'Fonebone'
person['age'] = 51
person['spouse'] = 'Edna'
person['children'] = ['Ralph', 'Betty', 'Joey']
person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}
print(person)
```

```
# short as one literal
person = {'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna',
          'children': ['Ralph', 'Betty', 'Joey'],
          'pets': {'dog': 'Fido', 'cat': 'Sox'}}
```

```
# alternative using the dict() function
person = dict(fname='Joe', lname='Fonebone', age=51, spouse='Edna',
        children=['Ralph', 'Betty', 'Joey'],
        pets={'dog': 'Fido', 'cat': 'Sox'})
```

# Control Structures

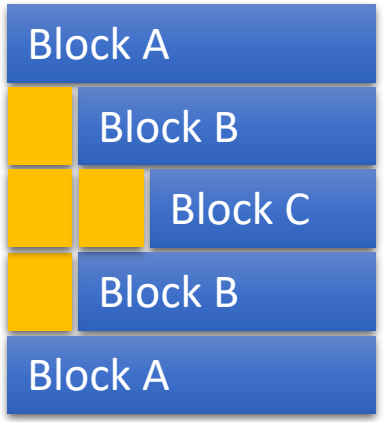
```
height = 3
width = 4

# one way if statement
if height != width:
    print("It's a rectangle!")

# two way if statement
if height != width:
    print("It's a rectangle!")
else:
    print("It's a square!")

# more than two ways
if height > width:
    print("It's a portrait rectangle!")
elif height < width:
    print("It's a landscape rectangle!")
else:
    print("It's a square!")
```

Blocks are definded by intending the code:



# The pass Statement

- Python doesn't allow empty blocks like in C#

```
if x < y:
    # implement later
else:
    print(x)
```

```
if (x < y)
{
    // implement later
}
else
{
    Console.WriteLine(x);
}
```

- If you have an empty block that you want to implement later, you must use the pass statement to fill the empty block

```
if x < y:
    # implement later
    pass
else:
    print(x)
```

# Conditional Expressions

```
raining = True

print("Let's go to the", 'beach' if not raining else 'school')
```

→Let's got to the school

## C#:

```
bool raining = true;

Console.WriteLine("Let's go to the " + (!raining ? "beach" : "school"));
```

# While Loop

```
secret = 1337
guess = -1
while guess != secret:
    guess = int(input("Guess the number: "))
print("You did it!")

# while loop with break and else
secret = 1337
guess = -1
while guess != secret:
    guess = int(input("Guess the number (0 = exit): "))
    if guess == 0:
        print('Game over!')
        break
else:
    print("You did it!")
```

# For Loop

- The for loop is similar to the foreach loop in C#

```
my_list = ['a', 'b', 'c']

for entry in my_list:
    print(entry)
```

=

```
List<string> myList = new List<string>(){ "a", "b", "c" };

foreach (string entry in myList)
{
    Console.WriteLine(entry);
}
```

- This for loop is similar to the for loop in C#

```
for i in range(1, 10, 1):
    print(i)
```

=

```
for (int i = 1; i < 10; i = i + 1)
{
    Console.WriteLine(i);
}
```

# Iterating Lists and Dictionaries

```
alphabet_list = ['alpha', 'bravo', 'charlie']
for entry in alphabet_list:
    print(entry)


teachers = {'huo': 'Robert Hufsky', 'woh': 'Harald Wolf'}
for key, name in teachers.items():
    print(key, '=', name)
```

# Functions

```
# known number of params
def sum(a, b):
    return a + b

print(sum(3,4))

# unknown number of parameters
def sum2(*params):
    sum = 0
    for param in params:
        sum += param
    return sum

print(sum2(1,2,3,4,5))
```

# Functions (2)

- Default Argument Values

```
def say(message, times=1):
    print(message * times)

say('Hello')
say('World', 5)
```

→

```
Hello
WorldWorldWorldWorldWorld
```

# Functions (3)

- Keyword Arguments

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

→

```
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

# Files

- Writing:

```
names = ['Hugo', 'Susi', 'Leo']

# open for 'w'riting
f = open('names.txt', 'w')
# write names to file
for name in names:
    # use write() and append \n:
    # f.write(name + '\n')
    # or use print() to write to the file f
    print(name, file=f)
# close the file
f.close()
```

# Files (2)

- Reading

```
# if no mode is specified,
# 'r'ead mode is assumed by default
f = open('names.txt')

line = f.readline()
while len(line) != 0:  # don't use line.count != 0
    # the `line` already has a \n
    # so print to the screen
    # without an additional \n
    print(line, end='')
    # read the next line
    line = f.readline()
# close the file
f.close()
```

# Exception Handling

```
f = None
try:
    f = open('names.txt','r')
    for line in f:
        print(line, end='')
except FileNotFoundError:
    print('File not found!')
except IOError:
    print('Error reading file!')
else:
    print('Reading finished')
finally:
    if f:
        f.close()
        print("File closed!")
    else:
        print('Nothing to close!')
```

# The ‘with’ statement

```
try:
    with open("names.txt") as f:
        for line in f:
            print(line, end='')
except FileNotFoundError:
    print('File not found!')
except IOError:
    print('Error reading file!')
else:
    print('Reading finished‘)

# no need for a finally block
```

# Custom Exceptions and Raising an Exception

```
# a special exception class for handling product exceptions
class ProductException(Exception):
    def __init__(self, message):
        self.message = message

# a negative price is forbidden
def __set_price(self, price):
    if price < 0:
        raise ProductException('Price must not be negative')
    self.__price = price
```

```
try:
    title = input('Title:')
    price = float(input('Price:'))
    quantity = int(input('Quantity:'))
    p = Product(title, price, quantity)
    my_cart.add(p)
    print_cart()
except ValueError:
    print('Invalid values!')
except ProductException as ex:
    print(ex.message)
```