1.1 Gugu-gaga level

1. Basic syntax
2. Variables and data typesLists, dictionaries, tuples, sets
3. Type casting, exceptions
4. Cycles and conditions
5. functions, built-in functions
6. Virtual environment

1.2 Advanced level

1. Classes and OOP
2. Encapsulation, inheritance, polymorphism
3. Methods
4. Decorators
5. Iterators and generators
6. RegEx - regular expressions
7. Data structures and algorithms:
    a. Arrays and linked lists
    b. Heaps, stacks and queues
    c. Hash tables
    d. Binary trees
    e. Recursion
    f. Sorting algorithms

2.1 – Django/FastAPI

2.2 - DBMS

1. PostgreSQL
2. MySQL

2.3 - HTML/CSS/Bootstrap

What to read

1. Super guide to HTML in eng
2. Handbook of HTML in eng
3. CSS Guide in RussianCSS reference book
4. HTML5 and CSS3 Guide
5. Bootstrap-4 Documentation
6. Bootstrap-5 Documentation

2.4 - What projects to complete to learn Django in practice

2.5 - DRF - Django REST Framework

1. What to read
    a. DRF - Manual in English

b. Books, articles, notes, cheat sheets on hashery #django #drt

2.6 + JS, Vue.js or React

1. What to read
    a. Vue.js Documentation in Russian
    b. React - Documentation in Russian

3 - Knowledge of the terminal and ability to work with Linux

2. Books, articles, notes, cheat sheets using the hashtag #linux
3. Cheap VPS servers for projects

4 - Docker

1. What to read
2. Docker tutorial in Russian

5 - Telegram bots

1. What to read
    a. Telegram APIs
    b. AIOGram documentation
    c. Fast start. Echo bot on AIOGram

BASE

To download python , you need to:

1. ***Download installing file from site https://www.python.org/downloads/***
2. ***After installation , you must choose comfort environment for you , example:***
    a. ***Visual Studio Code*** *(I prefer it)*
    b. ***PyCharm*** *(my teacher said its good)*
    c. ***Windows PowerShell*** *(I've never seen person who coding here)*
    d. ***Notepad*** *(No way bro)*
3. ***In future , or even on that stage of learning you'll need pip commands (like for installing "DLC" for Python , or to install python itself) :***
    a. ***To install pip , you must have Python , on your gadget , or write :*** `python get-pip.py`
    b. ***To further commands , you can look for python extensions on web sites , or for other needs , you can use*** `python -m pip help`
    c. ***Some other commands :***
        1) download – download packages.
        2) uninstall – uninstall packages.
        3) freeze – output installed packages in requirement format.

4) inspect – inspect python environment.
5) list – list installed packages.
6) show – show information about installed packages.
7) check – verify installed packages have compatible dependencies.
8) config – manage local or global configuration.
9) search – search PyPI for packages.

And if you want to downgrade your python version , write command : python -m pip install pip==18.1 (downgrade on version 18.1)

For upgrade , use : pip install –upgrade pip

1.1.1 Basic syntax

Print() – output , smth text on screen
X = 2 – create variable

1. You must do leave space before text in cycle or condition:
2. Also you need to put “:” after your condition to correct work of code

Example:

```
If x == 3:
    ->print(“x = 3”)
elif x > 3:
    ->print(“x>3”)
else:
    ->print("x<3”)
```

To create comment you must use # before text , that you want to write

Example:

```
#Creating tuple consisting passwords of users
```

Or you can turn more than 1 line in coment , by using

```
’’’’’’
Bruh
moment
’’’’’’
```

Operators:

1) + | 1+2 =3
2) – | 4-5 = -1

3) * |5*4 = 20
4) ** | 6**3 = 216 (6*6*6)
5) / | 5/4 = 1.25
6) // | 5//4 = 1
7) % | 6%4 = 2 , 5%4 =1
8) and | 5 and 4 == 3 (False)
9) or | 3 or 5 == 3 (True)
10) not | not False – True
11) in | for x in list
12) >= , <= | 5 >= 3 T , 6 <= 6 T
13) > , < | 6 > 7 F , 6 < 7 T
14) == | 5 == 5 , T
15) != | 5 != 4 , T

1.1.1.1 Print

Print command , have much of variants of usage , it cant be use like that:

```
x = 52
print(“Hello” , sep=’ ’)
print(“World” , sep=None)
print(x)
```

And like that :

```
x = 52
print(“Hello” + ”World” + x)
```

Or

```
print(“Hello World” + x) ,
```

Result, anywhere is (Hello World 52) . ***One, of important things here , is sep*** , which means separate , and it stands for what will be after your each printed value , so , if

sep = ” ” , after your word , wont be enter , and will just “ ” , same thing with ***sep = None*** , you may thought , it will be none , but it still add space after ***sep = “”*** , this one don’t add space after printed value , but you can add everything , what you have on your mind between “” – (“aboba”) , and it will write aboba , after printed .

And finally , ***sep="\n"*** , this add enter after your value , anyway , its not much useful , cause Python add enter by itself , without your wanting , you can don't write this sep , if you want to .

`print('ok','I'll','be fine',sep=None)` | result = "ok I'll be fine" , after each variable add space.

Another important thing is end , but it add values , after whole printed values , so

`print('ok','I'll','be fine',end=None)`

`print("Bye")` | result = "okI'llbefine Bye" , in one line , without enter

Final necessary mention : print(**f**"He said his name{name}") , **'f'** allows you to use variables inside of printing text , without + , or , .

1.1.2 Variables and data typesLists, dictionaries, tuples, sets

Simple Variables:

1) (***Numbers***) *Integers* = 5, *Float* = 5,5 , *Complex* = 4+3j
    1. x = 5
    2. x,y,z = 3,4,5 (means x = 3 , y = 4 , z = 5)
    3. x = y = z = 4 (means x = 4 , y = 4 , z = 5)
    4. print (x,y,z) (it prints 4 4 4)
    5. print (x+y+z) (it prints 12)
2) (***Text***) String = "Hi"
    1. Same to integers, but :
    2. print(x,y,z) , is same to print(x+y+z)
       `x = "Nuh, "`
       `y = "I'd"`
       `z = "win"`
       `print(x+y+z) – ("Nuh , I'd win")`
    3. You can take for example first letter , from your text variable:
       a = "Hello, world!"
       print(a[0]) (res is "H" , don't forget , counting starts from 0)
3) ***Boolean*** = True
4) ***none*** = None
5) ***Binary:*** *bytes , bytesarray , memoryview ,* x =2 , bytes_x = b"5" , x = bytearray(5) , x = memoryview(bytes(5))

6) ***ATTENTION*** **,** you can also turn one data type , to another , just with writing , what you want turn to , example :

```
x = 5  #5
s = str(x)  #”5”
b = bytes(x) #\x05
f = float(x)  #5
```

7) List : x = [”apple”,”pineapple”,”banana”]
8) Tuple : x = (”cucumber”,”cabage”,”cherry”)
9) Dictionary : {”favorite_food” : ”Tentacion”, ”name” : ”Drake”}
10) Set : {”Ken”,”Karson”}
11) ***To know what is data type*** , you need to write : print(type(x).

1.1.2.1 Strings

- You can print only first 3 letters from text , just using print(txt[:3]) , or from 1-st til 4-th letter print(txt[1:4])
- If you want modify string , you have such commands :
  - upper print(txt.upper()) – It turns all letters to capital
  - lower print(txt.lower()) – It turns all letters to lower
  - strip print(txt.strip()) – It removes white space , like “Hello ” , after print is – “Hello”
  - replace print(txt.replace(“H”,””)) – “Hello” -> “ello”
  - split print(txt.split(“,”)) – Hello, Mike -> (”Hello” , ”Mike”)
- Format : do you remember at print section , I writed , about f method , its really not for print , but its string method , like :
  - Name,age=”Jake”,”19”
    Txt = f”Okay {name} , you can be free”
    Message = “Hello {0} , you are turned {1} yesterday , here your {1} candles”.format(Name,age) – instead of {0} – Jake , {1} – 19
- You can style some text , with escape characters , like print(“we are \”guys\” so stay chill”) – result (we are “guys” so stay chill) , another escape characters
  - \’ – add single quote
  - \\ - backslash
  - \n – new line

- \r – Carriage return
- \t - Tab
- \b - Backspace
- \f – Form Feed
- \ooo – Octal value
- \xhh – Hex value

1.1.2.2 Lists

- You can know length of list by len() command len(kek) = 3 , kek =[“kj”,”jk”,”ih”]
- List items are – ***changeable***
- To take 1-st item from list , you must print x = kek[0] , and if you want last one , x = kek[len(kek)-1] , also you can take items from 2 to 5 , for example , x = kek[2:5] , how it was with strings , right ?
- There are some useful commands with lists :
  - insert | kek.insert(2,”ok”) , so “ok” adds on 2-nd index (3-rd item)
  - append | kek.append(“ok”) , “ok” adds in end of list
  - extend | kek.extend(lol) , list lol will be appended to kek
  - remove | you can remove item , kek.remove(“ok”)
  - pop | allows you to remove item with specified index , kek.pop(1) , but it also return element , so x = kek.pop(1) , so x will be equal to item on 1-st index from list
  - del | its like method pop , but it doesn’t return item
  - clear | clear whole list kek.clear[]
  - count | returns number of elements with specified value – list.count(“banana”)
- In future , we will discover looping methods , like for and while , so you can check lists with them , [print(x) for x in thislist] , `newlist = [x for x in thislist if "a" in x] - if item from thislist has ketter “a” in it , so it adds to newlist.`
- You can sort lists , with sort.() command , also sort(reversed = True) , will reverse sorted , so you can make list increasing and descending also . You can make your own sorting , if you want , I use only bubble sort , insert sort , or merge sorting .
- Reverse command , it works , like it looks like , reverse() , just reverse list
- Copy() method , just copies list , newlist = thislist.copy()

1.1.2.3 Tuple

- Tuple is unchangeable , so its how list , but more boring
- Instead of [] , tuple use () brackets
- Indexes in tuple , works like in lists
- You can turn list to tuple , with command tuple()
- To create tuple with single item , you must add comma after item thistuple=(“bro”,)
- Del word can delete whole tuple , but not indexed item in tuple del thistuple
- You can “unpack” tuple : thistuple = (”ok”,”meh”,”nuh”) , (o,m,n)=thistuple , so o = “ok” , m = “meh” , n = “nuh” , you can add asterisk before m , and remove n , so (o,*m) = thistuple , now o = ”ok”, m = [”meh”,”nuh”].
- You can add and multiply tuples:
    - tuple3 = tuple1 + tuple2
    - tuple2x = tuple1*2
- tuple methods:
    - count() | - works like in lists
    - index() | - returns values , on specified index

1.1.2.4 Set:

- Unlike of list and tuple , set ischangable , and it hasn’t order , so items always will be random queue .
- Set , cant have same values , it will merge it in one whole
- Set commands:
    - Add() – adds element to set
    - Update() – To add elements from one set to another
    - Clear() – clear set
    - Copy() – copy set
    - Difference() – x.difference(y) , returns items that contains x , but doesn’t contains y
    - Discard() – remove item with specified index in set
    - Pop() - remove item with specified index , and returns it
    - Remove() – remove item with specified name
    - Intersection() – reurns set , that contains item that have set x , and set y
    - Issubset() – returns whether another set contains this set or not True/False
    - Issupperset() – returns whether this set contains another or not , True/False

1.1.2.5 Dictionary

- Its ordered changeable , without duplicates
- thisdict={
- “age”: “23”,
- “name”:”John”
- }
- print(thisdict[“age”]) – 23
- Dictionary commands:
    - clear() | removes all elements from dict
    - copy() | return copy of dict
    - fromkeys() | return dict , with specified keys and value
    - get() | return value of specific key
    - items() | return a list containing tuple , for each key value pair
    - keys() | return list containing every key
    - pop() | remove and return the element with specified key
    - popitem() | remove and return last inserted key-value pair
    - setdefault() | return value of specified key , if no key , it creates key with value , if wrong value , it returns value , of key .
    - update() | add to dictionary specified key-value pairs
    - values() | return list with all value in the dictionary
    - del | can delete item from dict , or whole dict, so you even cant print it
-

1.1.2.6 Module (Its just demo version) :

First module , with what you can play around , and create your first interesting code , its ***random ,*** you need to import it to Python , so after that you can use it , example :

Import random as rn

1. x = rn.randrange(1,9,2) 1 – min num , 9 – max num , 2 – step (so you can inspect only 1 , 3 , 5 , 7) YOU CANT INSPECT MAXIMUM VALUE IN RANDRANGE , CAUSE IT 1<=x<9
2. x = rn.randint(1,9) 1 – min num , 9 – max num , 1<=x<=9 , its ok
3. k = “dsafecwevr”
   x = rn.choice(k) (x = 1 letter from k) (it takes random item from tuple also , or cortege,)

1.1.3 Type casting, exceptions

- To turn one type to other, you need convert it like:
- x = 5
- x_s = str(x)
- #x_s = “5” ITS CALLED EXPLICT
- BUT you can’t convert string to int
- Implict type casting , its:
- x = 5
- y = 2.0
- z = x+y
- #Z automatically converts to float
- EXCEPTIONS :
- Idk , what about this need to be , so this part I take from my head :
    - You can avoid some code mistakes , by try , except , except methods , for example

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
    print("Result:", result)
except ValueError as ve:
    print("Invalid input. Please enter an integer.")
except ZeroDivisionError as zde:
    print("Error: Division by zero is not allowed.")
finally:
    print("This block executes no matter what.")
```

    - Here , we see how except helps us to avoid ValueError or ZeroDivisionError , without stoping compilator
    - You can also name error , with raise method , which allows you to , for example , give name to ValueError , and when we type print(ve) , there are text for us

1.1.4 Cycles and conditions

1. If | so commonly used operator , if x > 3 and x < 9: print(“Ok”)
2. Else | you have to know it too , print(“Ok”) if x > y else print(“Nuh”)
3. Elif | its not for C# users , when I firstly have seen it , I was in shock , its just works how if , but you must place it after if ,
    a. if x > y:

b. print("Ok")
c. elif x == y:
d. print("Ok")
e. else
f. print("nuh")

4. While | How it sounds , how it supposed to work :
   a. while i > 6:
      i. i += 1
   b. #So when i will be 6 , while loop will stopped
   c. Also to stop while loop , you can use break method , so just putting word break , when loop will come to this word , it will be stopped
   d. If we put continue word , into loop , we will skip further lines of code , if they are exist , and come
5. For | It easy to get , how it works, just:
   a. for i in range(1,6,3):
      i. print(i)
      ii. #result 1 , 4
   b. Break also works in for
   c. If loop is empty , you can use ***pass*** to avoid error

1.1.5 functions, built-in functions

- To name function , you must write def before , name , so it looks like:
  - def abs_func(num,pown):
    - num = abs(num)
    - num = pow(num,pown)
    - return num
- You can use asterisk before variable:
  - def nam_func(*names):
    - print(kids[2])
  - nam_func("Frank","Toby","Tom")
- You can indicate variables like:
  - def nam_func(name1,name2,name3):
    - print(kids[2])
  - nam_func(name1 = "Frank","Toby","Tom")
- Another method with asterisks , but no its 2 of it:
  - def nam_func(**nam):
    - print(nam[1])

- nam_func(0 = “Tom” , 1 = “Bob”)

- You can also put lists in func , and do default arguments , that will be used , if you wont type any variable in nam_func() , it looks like:
  - def nam_func(nam=”Bob”)
- You can use pass method , to avoid errors during the function
- And last but not least , its function recursion , you can call func , inside of itself :

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

print("\n\nRecursion Example Results")
tri_recursion(6)
```

  - #Result :
    - 1
    - 3
    - 6
    - 10
    - 15
    - 21

- Lambda function , it’s our build-in functions , that we can use on our point of view:
  - x = lambda a : a + 10
  - print(x(5)) #result is 15
  - x = lambda a , b : a*b
  - print(x(5,6)) #result is 30 , you can add as many as you want variables to lambda func
- You can use lambda function , inside of another function , for better and more economic code :
  - Def m_func(n):
    - Return lambda a : a*n
  - mydoub = m_func(2) #here we give n=2
  - print(mydoub(11)) #result is 22 , we designate a variable here

1.1.6 Virtual environment

Just imagine situation , when one python project needs version of library 1.9 and another one – 2.1 , so you cant work at both of them , cause if you download both versions on your PC , they will conflict , so here on help come virtual environment , so you can work at different projects , with its own dependencies , so allows you to use different versions . But how create virtual environment ?

1. First of all , you must find folder where you plan to create virtual environment
2. Before it , you must install `pip install virtualenv`
3. After it , you open cmd , in it , so it must write
    a. C:\MyPythonProject>
4. Now you type command : python -m venv myenv
5. To activate your virtual environment you need to type command :
    a. myenv\Scripts\activate #instead of myenv , can be your name
6. Ok , to deactivate , just type deactivate
7. If you in your virtual environment , you can install different features here , like another version of Django , or downgrade Python ver , by commands pip install Django
8. But to downgrade Python version , you must install your wanted version from https://www.python.org/downloads site , and then , Type into the command prompt: virtualenv \path\to\env -p \path\to\python_install.exe, whereas \path\to\env shall be the path where your virtual environment is going to be and \path\to\python_install.exe the one where your freshly (presumably) installed Python version resides.

Resourses

https://stackoverflow.com/questions/57150426/what-is-printf

https://www.w3schools.com/python

PythonCheatSheet.allinpython.com

https://www.geeksforgeeks.org/python-random-module/