Dervis Mehmed Barutcu
Softtech - Stajyer

Python – Basics

What is python?

Python is an interpreted programming language. This means it can be executed as soon as it is written.

How to install Python 3 and PyCharm?

For all operating systems, python can be installed from its website. To check whether python is properly installed, a user should check the version of the python.

For Windows, the user should open the command line and should write "python --version" to see version of the python.

For Mac OS and Linux operating systems, the user should open the terminal and should write "python3 --version" or "python --version" to verify the installation.

In order to install PyCharm, the user should download the installation folder from Jetbrains website according to operation system, and install the PyCharm. After the installation is complete, the user need to set the interpreter.

How python works?

Python is a dynamic, interpreted (bytecode-compiled) language. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible, and you lose the compile-time type checking of the source code. Python tracks the types of all values at runtime and flags code that does not make sense as it runs.

Print():function prints the specified message to the screen.

Syntax: print(object(s), sep=separator, end=end, file=file, flush=flush)

object(s): Any object or objects separated by commas.

sep: Optional parameter. If there are more than one objects, it specifies how to separate the objects. Default is ' '.

end: Optional parameter. Specify what to print at the end. Default is '\n'.

file: Optional parameter. An object with a write method. Default is sys.stdout.

flush: Optional parameter. Boolean value for if the output is flushed or buffed. Default value is False.

input(): the user can enter inputs to the program. However, input() function converts the inputs into a string whether it is integer, string etc. you can later convert the input any type by type casting it.

Dervis Mehmed Barutcu
Softtech - Stajyer

Data Types in Python

Python has the following data types by default.

Text type:	str
Numeric types:	int, float, complex
Sequence types:	list, tuple, range
Mapping Type:	dict
Set types:	set, frozenset
Boolean type:	bool
Binary types:	bytes, bytearray, memoryview

Type Casting

Type casting is used when the developer wants to specify a type on to a variable. The followings are the in-built functions used for type casting:

int(): constructs an integer number from an integer literal, a float literal by removing all decimals, or a string literal providing the string represents a whole number.

float(): constructs a float number from an integer literal, a float literal or a string literal.

str(): constructs a string from a wide variety of data types, including strings, integer literals and float literals.

Examples:

x = int(1) -> x is 1	x = float(1) -> x is 1.0	x = str("s1") -> x is 's1'
x = int(3.4) -> x is 3	x = float(1.6) -> x is 1.6	x = str(2) -> x is '2'
x = int("3") -> x is 3	x = float("1.6") -> x is 1.6	x = str(2.5) -> x is '2.5'
	x = float("3") -> x is 3.0	

Format(): formats the specified value(s) and insert them inside the string's placeholder. Placeholders are defined using curly brackets: {}, which can be identified using named indexes {price}, numbered indexes {0}, or empty {}.

Example for named indexes, numbered indexes and empty placeholders:

- txt = "My name is {name}, and I am {age}".format(name = "Dervis", age = 26)
  print(txt) # My name is Dervis, and I am 26
- txt = "My name is {0}, and I am {1}".format("Dervis", 26)
  print(txt) # My name is Dervis, and I am 26
- txt = "My name is { }, and I am { }".format("Dervis", 26)
  print(txt) # My name is Dervis, and I am 26

Some of the formatting types are:

- ":^" center aligns the result within the available space.
- ":," use comma as a thousand separator
- ":f" fix point number format

Condition

Python supports the logical conditions from mathematics.

- Equals:	a == b
- Not Equals:	a != b
- Less than:	a < b
- Less than or equal to:	a <= b
- Greater than:	a > b
- Greater than or equal to:	a >= b

if…else

Python relies on indentation to define scope in the code. For example:

```
a = 33

b = 34

if a > b:

        print ("a is greater than b")

elif a < b:

        print("b is greater than a")

else:

        print("a is equal to b")
```

The output of the above code is "b is greater than a". First condition is false because a is less than b, so the second condition is tried and this condition is true.

If there are more than one conditions to be checked, "and" or "or" operators can be used.

If conditions are combined with "and", and all the conditions are true, if statement returns true.

If conditions are combined with "or", and at least one condition is true among conditions, if statement returns true.

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations.

- Addition	+	x + y
- Subtraction	-	x – y

-	Multiplication	*	x * y
-	Division	/	x / y
-	Modulus	%	x % y
-	Exponentiation	**	x ** y
-	Floor division	//	x // y # rounds the result down to the nearest whole number

While Loop

While loop can execute the statements as long as a condition is true. For Example:

Code	Output
`i = 1`	1
`while i < 6:`	2
`    print(i)`	3
`    i += 1`	4
	5

For Loop

For loop is used for iterating over a sequence. For example looping through an array:

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

Another example for loop:

Code	Output
`for x in "banana":`	b
`  print(x)`	a
	n
	a
	n
	a

x takes each letter and is printed until the string finishes.

Range, Pass, Break and Continue

range(start, stop, step) function returns a sequence of numbers started from start until stop. Starting from 0 by default and steps by 1. For example:

Code	Output
`x = range(3, 20, 2)`	3
`for n in x:`	5
`    print(n)`	7
	9
	11
	13

	15 17 19

pass is a placeholder and when pass statement is executed nothing happens, but you avoid getting an error when empty code is not allowed. Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

continue keyword is used to skip current iteration in a loop and continues to the next iteration.

break keyword is used to end the loop at the current iteration.

Arrays

Python does not have built-in support for arrays but Lists can be used as arrays.

Lists

Lists are used to store multiple items in a single variable. They are created using square brackets.

Code	Output

list = ["apple", "banana", "cherry"]
print(list)
	['apple', 'banana', 'cherry']

List items can be accessed by their index values. The first item's index is zero. Index value is increment one by one. List items can also be accessed by specifying range:

Code	Output

list = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(list[2:5])
	['cherry', 'orange', 'kiwi']

The code above will return the items from index 2 to 5.

Changing items

Changing the item in the list is done by referring the index number.

Code	Output

list = ["apple", "banana", "cherry"]
list [1] = "blackcurrant"
print(list)
	['apple', 'blackcurrant', 'cherry']

Although there are no blackcurrant in the list, the item that has position 1, is changed with the “blackcurrant”.

In python, to change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Code	Output

list = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
list [1:3] = ["blackcurrant", "watermelon"]
print(list)
	['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']

Adding items

- append() function is to add an item to the end of the list.
- insert() function is to insert a list item at a specified index.

Code	Output

list = ["apple", "banana", "cherry"]
list.insert(1, "orange")
print(list)
	['apple', 'orange', 'banana', 'cherry']

- extend() function is to append elements from one list to the current list.

Removing items

- remove() function removes the specified item.

Code	Output

list = ["apple", "banana", "cherry"]
list.remove("banana")
print(list)
	['apple', 'cherry']

- pop() function removes the specified index. If no index is specified, pop() removes the last item.
- clear() function empties the list.

Other list methods are:

- copy():	returns a copy of the list.
- count():	returns the number of elements with the specified value.
- index():	returns the index of the element with the specified value.
- reverse():	reverses the order of the list.
- sort():	sorts the list.

Reading File

open() function is for working with files. It takes two parameters; filename and mode. There are four different modes for opening a file:

- “r”: Read. Default value. Opens a file for reading, gives error if the file does not exist.
- “a”: Append. Opens a file for appending, creates the file if it does not exist.
- “w”: Write. Opens a file for writing, creates the file if it does not exist.
- “x”: Create. Creates the specified file, returns an error if the file exists.
- “t”: Text. Default value. Text mode.
- “b”: Binary. Binary mode (e.g. images).

Code	**Output**
f = open("demofile.txt", "r")	Hello
print(f.read())	World!

read() function reads the whole text file and returns it. It can also be specified how many characters want to be returned.

readline() function reads and returns one line.

Whole file can be read by looping the lines.

Code	**Output**
f = open("demofile.txt", "r")	Hello
for x in f:	World!
print(x)	

Enumerate()

enumerate() is a built-in function that adds a counter to an iterable and returns it as enumerate object. This enumerate object can be used in for loops or converted into a list using list() function.

Code	**Output**
list = ["eat","sleep","repeat"]	(0, 'eat')
	(1, 'sleep')
for ele in enumerate(list):	(2, 'repeat')
print (ele)	

String Methods

- replace(): replaces all occurrences of the specified phrase with another specified phrase.
  - Syntax: str.replace(oldvalue, newvalue, count)
- split(): splits a string into a list.
  - Syntax: str.split(separator, maxsplit)
    - maxsplit specifies how many splits to do
- rsplit(): splits a string into a list, starting from the right.
  - Syntax: str.rsplit(separator, maxsplit)
- splitlines(): splits a string into a list. The splitting is done at line breaks.
  - Syntax: str.splitlines()
- lower(): returns a string where all characters are lower case.
  - Syntax: str.lower()
- upper(): returns a string where all characters are upper case.
  - Syntax: str.upper()
- islower(): returns True if all the characters are in lower case, otherwise False.
  - Syntax: str.islower()
- isupper(): returns True if all the characters are in upper case, otherwise False.
  - Syntax: str.isupper()

- capitalize(): returns a string where the first character is upper case.
  - Syntax: str.capitalize()
- count(): returns the number of times a specified value appears in the string.
  - Syntax: str.count(value, start, end*)*
    - value: a String
    - start: an Integer, starting position
    - end: an Integer, ending position
- find(): finds the first occurrence of the specified value. If the value is not found, returns -1.
  - Syntax: str.find(value, start, end)
- isdigit(): returns True if all the characters are digits, otherwise False.
  - Syntax: str.isdigit()
- isnumeric(): returns True if all the characters are numeric (0-9), otherwise False.
  - Syntax: str.isnumeric()
- isalpha(): returns True if all the characters are alphabet letters (a-z).
  - Syntax: str.isalpha()
- isspace(): returns True if all the characters in a string are whitespaces, otherwise False.
  - Syntax: str.isspace()

Dervis Mehmed Barutcu
Softtech - Stajyer

Example Projects

Calculator

```
q = "n"
while(q != "y"):
    x = int(input("1: "))
    y = int(input("2: "))
    z = input("operator: ")

    if z == "+":
        print(x + y)
    elif z == "-":
        print(x - y)
    elif z == "*":
        print(x * y)
    else:
        print(x / y)
    q = input("Quit? (y for yes)")
```

It takes three inputs. Two input for the numbers to be calculated and third input is for the operation. The code will run until the user enters “y”.

Counting Characters and Spaces in a File

```
f = open("demofile.txt", "r")

numOfChar = 0
numOfSpace= 0

for line in f:                               # to iterate each lane
        line = line.strip("\n")              # strip from “\n” to exclude it
        numOfSpace+= line.count(" ")         # to count how many spaces
        numOfChar += len(line)               # to count how many characters

print("Number of characters:",numOfChar, "\nnumber of spaces:", numOfSpace)
f.close()
```

The code above opens the code for reading. Then, iterate each line one by one with the for loop. Inside of the loop, it splits the line from “\n” character to count letters. Count the spaces in each line and sum them iteratively, and from length of the line, summation is done to find total number of the characters.