PYTHON NOTES

FUNCTIONS: A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

* *Arguments*- it is the actual piece of data that is passed to the function when it is called.
* *Parameters*- it is the name of the data that is used inside the function to refer to it and to do things with it.
* In the below code “name” is the *parameter* and “Thapasya” is the *argument*.

def greet(name):  
 print(f"hi {name}")  
 print(f"hello {name}")  
 print(f"welcome back {name}")  
  
  
greet("Thapasya")

* functions that allows for a multiple inputs – and those arguments are called as ***positional arguments***.
* *location* = “India” and *name* = “Thapasya” are the *keyword arguments*. In the below lines.
* def greet\_with(name, location):  
   print(f"Hello {name}")  
   print(f"What is it like in {location}")  
    
    
  greet\_with(location="India", name="Thapasya")  
  # or  
  # greet\_with("THapasya", "India")

>>***len():*** to find out the number of letters in a character.

Eg- num\_of\_letters = len(“Thapasya”)

print(num\_of\_letters) - answer = 8

print(len(int(“apple”))) – answer = apple

>>***type():*** to check the type of data we use.it is a data type checking function.

Eg-print(type(num char)) - it’s a string data type

>>Type conversion / Type casting: it is the process of change of the data types.

\*str() function is used to convert other data types into a string so that it can be concatenated.

Eg- num = 3

print(“number is ”+str(num))

In python commas (,) are described as underscore(\_) but the compiler does not care about it, its for our understanding .eg- print(3\_200)

>>***split():*** The string manipulation function in Python used to break down a bigger string into several smaller strings is called the split() function in Python. The split() function returns the strings as a list.

>> ***round():***The round function in Python is used for rounding off numbers up to a specified number of digits after the decimal point. Its symbol is (//)

>>***higher order functions***- a function that can work with other functions, at that time the function that is within the other function doesn’t need a parenthesis.

Calculate function is the HIGHER ORDER FUNCTION below-

def add(n1, n2):  
 return n1 + n2  
  
  
def sub(n1, n2):  
 return n1 - n2  
  
  
def mul(n1, n2):  
 return n1 \* n2  
  
  
def div(n1, n2):  
 return n1 / n2  
  
  
def calculate(n1, n2, func):  
 return func(n1, n2)  
  
  
result = calculate(2, 3, mul)  
print(result)

***Lists***:

* *append*- to add a word to the end of the list.
* *extend*- to add a sentence to the end of the list.

To change the index of the list we need to:

fruits = ["apple", "banana", "mango", "grapes"]  
fruits[1] = "water"  
print(fruits)

***NESTED LIST:***

fruits = ["Strawberries", "Nectarines", "Apples", "Grapes", "Peaches", "Cherries", "Pears"]  
vegetables = ["Spinach", "Kale", "Tomatoes", "Celery", "Potatoes"]  
  
dirty\_dozen = [fruits, vegetables]

print(dirty\_dozen)

***f-string:***

So, to convert various data type into a convenient or specific Data type according to your statement is done by an *f-string* .

Eg- name = "Thapasya"  
date\_of\_birth = 10\_9\_2009  
height = 1.63  
wishes = "happy birthday to you"  
boolean = True  
  
print(f"hello {name} {wishes} \nthank you\n its {date\_of\_birth} and its {boolean} that my height is {height}")

PEDMAS:

Note : modules % - gives the remainder.

Divide / - gives the quotient

p- Parentheses or Brackets ()

e- Exponents \*\*

d- Divide /

m- Multiply \*

a- Addition +

s- Subtraction –

Note: the calculation always goes from left to right.

Eg- print(3 \* 3 + 3 / 3 - 3) - answer=7

print(3 \* (3 + 3) / 3 - 3)) - answer=3

Short Hand Operators:

Sum += 1

Diff -= 1

Prod \*= 2

Quot /= 2

Remainder %= 2

Comparison Operators:

* - greater than

< - less than

>= - greater than equal to

<= - less than equal to

== - double equal to (to check if both the values are equal) – assignment operator.

!= - not equal to

Data Types:

*Integer*: stores the int values.

*Float*: stores the decimal values.

*String*: stores the set of characters.

*Boolean*: it gives out True or False.

print("Day 1 - String Manipulation")  
print("String Concatenation is done with the '+' sign.")  
print('example: print("Hello" +"world")')  
print("New lines can be created with a backslash and n. - /n")

|  |  |
| --- | --- |
| print | input |
| print function displays the given message on the screen. | input function accepts given data. |

|  |  |
| --- | --- |
| randint | len() |
| The colors list contains six colors, and the randint() function is used to generate a random index between 0 and 5 (inclusive) to pick a random color from the list. | The len() function is used to get the length of the list . it counts from 1 . |

* To find the random numbers in decimal data type – random\_float = random()
* random(1, 5) – it gives the random numbers between 1.00000… to 4.999999….. but not 5.
* To find the random number between 1 to 5 – random\_integer = randint()

>>***class inheritance***- the process of inheriting the behavior and appearance from an existing class.

So, to inherit a class-

class Animal:  
 def \_\_init\_\_(self):  
 self.num\_eyes = 2  
  
 def breath(self):  
 print("inhale, exhale")  
  
  
class Fish(Animal):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 def breath(self):  
 # to inherit the method  
 super().breath()  
 print("doing this in water")  
  
 def swim(self):  
 print("moving in water")  
  
  
nemo = Fish()  
nemo.swim()  
nemo.breath()  
print(nemo.num\_eyes)

>>***Slicing*-** String slicing in Python is about obtaining a sub-string from the given string by slicing it respectively from start to end.

Eg-

Piano\_keys = [“a”, “b”, “c”, “d”, “e”, “f”, “g”]

#to slice it from position c to e i.e 2 to 5:

Print(piano\_keys[2 : 5])

# to slice from position 2 to the end of the list:

Print(piano\_keys[2:])

# to slice from the starting position to the position 5:

Print(piano\_keys[:5])

# slicing between 2 positions by incrementing it by a certain number:

Print(piano\_keys[2:5:2]) - piano\_keys[start:stop:step]

# to slice everything by 2 phases:

Print(piano\_keys[::2])

# to reverse the order of the list:

Print(piano\_keys[::-1])

Note: all these methods can also be used in a tuple.

Eg – piano\_tuple = [“do”, “re”, “mi”, “fa”, “sa”, “la”, “ti” ]

Print(piano\_tuple[2:1])

>>***Object Oriented Programming Language(OOP)-*** it splits a larger task into a smaller number of modules.

OOP has :

* *Attributes* – they are the things that the object has.
* *Methods* – they are the things that the object does.

When a function is added to an object then it is called a method.

**Classes** are the blueprint to create objects.

>>**Constructor-** it is the part of the blueprint that allows us to specify what should happen when our object is constructed. It is also called as an initialize.

def\_\_init\_\_(self): - creates starting value to the attributes.

>>**Concatenating** means obtaining a new string that contains both of the original strings or by combining or adding 2 or more strings together.

In concatenation you can only concatenate 2 string but not int values.

Eg- n = 3

Print(“your name has”+n+”characters”) – this line gives Typing error.

>>**Subscript** means pulling out the single character from a string.

Eg- print(“thapasya”[3])

Print(“hello”[4])

>> **Dictionary:** a python dictionary is a data structure that stores the values in key : value pairs. It has elements that are identified by their key. Dictionary = {key : value}

programming\_dictionary = {  
 "Bug": "An error in a program that prevents the program from running as expected.",  
 "Function": "A piece of code that you can easily call over and over again."  
}  
print(programming\_dictionary)

* to add the latest item in your code-

programming\_dictionary["loop"] = "The action of doing something over and over."  
print(programming\_dictionary)

* to wipe an existing dictionary-

programming\_dictionary = {}  
print(programming\_dictionary)

* to edit a pre-existing item in your dictionary-

programming\_dictionary["Bug"] = "Insect around the flowers"  
print(programming\_dictionary)

* to loop though a dictionary-

for key in programming\_dictionary:  
 print(key) # only to print the key  
 print(programming\_dictionary[key]) # to print key as well as the value

capitals = {  
 "India": "New Delhi",  
 "France": "Paris"  
}

* Nesting a list inside a dictionary-

Travel\_log = {  
 "India": ["New Delhi", "Gujarat", "Telanga"],  
 "France": ["Paris", "little", "Puppy"]  
}

* Nesting a dictionary inside a dictionary-

Travel\_logo = {  
 "India": {"cities\_visited": ["New Delhi", "Gujarat", "Telangana"], "Total\_visits": 5},  
 "France": {"cities\_visited": ["Paris", "little", "Puppy"], "Total\_visits": 0}  
}

* Nesting a dictionary inside a list-

Travel\_logs = [  
 {  
 "country": "India",  
 "cities\_visited": ["New Delhi", "Gujarat", "Telangana"],  
 "Total\_visits": 5  
 },  
 {  
 "country": " France",  
 "cities\_visited": ["Paris", "little", "Puppy"],  
 "Total\_visits": 0  
 }  
]

>>**Tuple:** it is one of the 4 built-in data types in python used to store the collections of data, the other 3 are:

1. List
2. Set
3. Dictionary

All with different qualities and useage.

>> **Docstrings:** it converts the user defined function into a computer defined and then it gets populated in the docstring. Or

It creates definitions to the user defined functions. Or Python docstrings are the string literals that appear right after the definition of the function, method, class, or module.

Eg: len(name)

# >>  
# builtins  
# [def len(\_\_obj: Sized) -> int  
# Return the number of items in a container.  
# `len(\_\_obj)` on docs. python. org\]

The above lines will be displayed when the cursor is placed on the built-in functions. So, the docstrings will convert and display the similar type of lines to the user defined functions.

def function\_output(f\_name, l\_name):  
 *"""Takes the first and the last name and then formats it to a title case version of that name"""* if f\_name == "" or l\_name == "":  
 return "invalid input"  
 formated\_f\_name = f\_name.title()  
 formated\_l\_name = l\_name.title()

>> **File Paths:**

* Absolute file path - it starts with a back slash (/)
* Relative file path - it says to check first in the current open file and not to begin from start.

It starts with a

./ - to move one step forward.

../ - to move one step backward.

file = open("my\_file.txt")  
# contents = file.read()  
# print(contents)  
# file.close()  
#  
# # or , instead of opening and closing we can use a while loop  
#  
# with open("my\_file.txt") as file:  
# contents = file.read()  
# print(contents)  
#  
#  
# # to write -  
# # mode="a" - to append new text to an existing code  
# # "r" - it goes to read only mode  
# # "w" - to write new text . it erases the old text and writes the new one  
#  
# with open("my\_file.txt", mode="a") as file:  
# file.write("\nnew text.")  
#  
# # if you want to write a new text in a file that doesn't exist then the python itself creates a new file  
#  
#  
  
with open("C:/Users/Siddhartha/workspace\_python/pythonProject/newTextFile.txt") as file:  
 contents = file.read()  
 print(contents)  
  
with open("/Users/Siddhartha/workspace\_python/pythonProject/newTextFile.txt", mode="a") as file:  
 file.write("welcome to my python project")

>> **Scopes:**

* Local scope- it exists only within the function. Or The variables that can be used within the function only.
* Global scope- these variables are accessible both inside and outside the function.
* Global constancies- it stores the values that we never want to change in future circumstances and it is in all-caps.

Links:

1. reeborge.co
2. patorjk.com
3. ASCII ART – ascii.co.uk/art
4. Turtle graphic documentation
5. Basic git commands
6. Github.com/ThapasyaN - to enter into my git account