basics-of-python

March 30, 2024

1 WEEK 1 : BASICS OF PYTHON

1.1 NUMBERS

1.1.1 The primary Python types for numbers are int and float.To put it simply, integers are whole numbers such as 31, 50, and 71.

1.1.2 Conversely, fractional numbers like 3.14, 2.867, and 76.88887 are known as floats.

Let's verify their data types by adding a few arbitrary numerical values.

[ ]:
```
type(30)
```

[ ]:
```
int
```

[ ]:
```
type(3.14)
```

[ ]:
```
float
```

The float and numeric values can be kept in a variable. Furthermore, there's no need to initialise the variable with any kind of data type.

[ ]:
```
a = 10
type(a)
```

[ ]:
```
int
```

[ ]:
```
num = 10.2543
type(num)
```

[ ]:
```
float
```

able to apply simple mathematical operations

[ ]:
```
# Addition
8+3
```

[ ]:
```
11
```

[ ]:
```
#Substraction
8-3
```

[ ]: 5

[ ]:
```
#Multiplication
8*3
```

[ ]: 24

[ ]:
```
#Division
8/3
```

[ ]: 2.6666666666666665

[ ]:
```
#Power
8**3
```

[ ]: 512

[ ]:
```
#parantheis precedence
num1 = (15+62)%12
num2 = 15 + 62 % 12
print(num1)
print(num2)
```

5
17

There's a special operator called modulus which returns the remainder after integer di

[ ]:
```
#Modulus
8%3
```

[ ]: 2

Another way to execute the arithmetic operation is to initialise it into a few variables and then work with it.

[ ]:
```
num1 = 15
num2 = 15
sum = num1+num2
print(sum)
```

30

[ ]:
```
num1 = 15
num1 = num1 + 1
print(num1)
```

```
16
```

The above code can be simplified and written as

[ ]:
```
num1 = 15
num1 = num1 + 1
print(num1)
```

```
16
```

1.2 STRINGS

1.2.1 In Python, a string is a type of data structure that holds a string of characters. Since it is an immutable data type, you are unable to alter a string after you have created it.

1.2.2 In python you can also add numeric values or symbols in a quotes of the string.

1.2.3 Python is able to recognize single quotes and double quoted at the beginning and the ending of the string

[ ]:
```
"Hello World"
```

[ ]:
```
'Hello World'
```

Here it is able to recognize the question mark and comma as a string

[ ]:
```
"Hey, How are you ?"
```

[ ]:
```
'Hey, How are you ?'
```

We are able to perform the join operations of string i.e concatenation of the strings as well by storing it into the variables.

[ ]:
```
a = 'first'
b = 'last'
a+b
```

[ ]:
```
'firstlast'
```

If you want a space between the two words, you can simply add a space in the string instead of the character.

[ ]:
```
a = 'first '
b = 'last'
a+b
```

[ ]:
```
'first last'
```

There are different string method you can use such as upper(),lower(), replace(),count()

upper() changes the whole string into uppercase letters.

[ ]:
```
Str = 'Hello'
Str.upper()
```

[ ]:
```
'HELLO'
```

lower() changes the whole string into lowercase letters.

[ ]:
```
Str = 'HELLO'
Str.lower()
```

[ ]:
```
'hello'
```

replace() is a function with the help of which we are able to replace any character with another character

[ ]:
```
Str = 'goo'
Str.replace('o','a')
```

[ ]:
```
'gaa'
```

count() function let's us know the exact count of the appearance of any word in a string

[ ]:
```
num_list = ['one','two','one','two','two']
num_list.count('two')
```

[ ]:
```
3
```

format() method is been used for add some word or strings into a larger string at any particular index.

[ ]:
```
Str = "{0}, How are {1}?".format("Hey!!","you")
Str
```

[ ]:
```
'Hey!!, How are you?'
```

2 BOOLEANS

2.0.1 Boolean values are simply the True or False

2.0.2 With the help of booloeans, we can verify the intergers as well as the strings are correct or not, or else we can compare to

integer

[ ]:
```
100==100
```

[ ]:
```
True
```

[ ]:
```
10==11
```

[ ]:
```
False
```

we can also do with variables

[ ]:
```
a = 10
b = 10
a==b
```

[ ]:
```
True
```

string

[ ]:
```
"hey"=="hey"
```

[ ]:
```
True
```

[ ]:
```
a = "hello"
b = "hey"
a==b
```

[ ]:
```
False
```

using with Comparison operators like - < , > , <= , >=

[ ]:
```
10<11
```

[ ]:
```
True
```

[ ]:
```
12>14
```

[ ]:
```
False
```

[ ]:
```
10<=10
```

[ ]:
```
True
```

[ ]:
```
12>=11
```

[ ]:
```
True
```

we can also compare the strings.

[ ]:
```
"hey">="hey"
```

[ ]:
```
True
```

[ ]:
```
"hey"<="hello"
```

[ ]:
```
False
```

[ ]:

3 COLLECTIONS

3.0.1 Collection is nothing but a data structure where we can store multiple values.

3.0.2 Collections include

- List
- Dictionary

3.0.3 List

list is a container which holds multiples values.

list can accept multiple values of different data type

[ ]:
```
Name = ['Amit','Aman','Vineet','Saniya']
Name
```

[ ]:
```
['Amit', 'Aman', 'Vineet', 'Saniya']
```

To fetch any particular element from the list, we need to get its associated index. Remember, the index value of the list starts with 0, never with 1.

For the fetching, we just need to write the list name along with the square brackets in which the index value will be added to get the desired element.

[ ]:
```
#lets fetch the name vineet as he is at the index 2 starting the with zero.

Name[2]
```

[ ]:
```
'Vineet'
```

To keep track of the length of the list, i.e., to see how many elements have been added, the len() function has been used.

[ ]:
```
len(Name)
```

[ ]:
```
4
```

The append() function is used for adding an element to the last of the list.

[ ]:
```
Name.append("Hina")
```

[ ]:
```
Name
```

[ ]:
```
['Amit', 'Aman', 'Vineet', 'Saniya', 'Hina']
```

for checking whether the element is present or not in the list, without any loop

[ ]:
```
'Hina' in Name
```

[ ]:
```
True
```

3.0.4 Dictionaries

A dictionary optimises element lookups. It uses a key or value instead of numbers as placeholders.

[1]:
```
#lets create a dictionary
Dict = {'Banana':'yellow','apple':'red','orange':'orange'}
```

[2]:
```
Dict
```

[2]:
```
{'Banana': 'yellow', 'apple': 'red', 'orange': 'orange'}
```

If we ever want to fetch any value of the key, we just need to write the dictionary name and then continue with the square bracket in which the key is passed.

[3]:
```
Dict['Banana']
```

[3]:
```
'yellow'
```

A dictionary can also work with the numbers.

[5]:
```
dict = {'one':1,'two':2}
dict
```

[5]:
```
{'one': 1, 'two': 2}
```

[7]:
```
dict['one']
```

[7]:
```
1
```

There are some functions that are mostly used in the dictionary. They are keys() and values().

Keys(): It is a function that gives all the keys present in the dictionary.

[8]:
```
dict.keys()
```

[8]:
```
dict_keys(['one', 'two'])
```

values(): It is a function that gives all the values present in the dictionary.

[9]:
```
dict.values()
```

[9]:
```
dict_values([1, 2])
```

4 CONTROL STATEMENTS

4.0.1 Control statements are nothing but a condition checker, i.e., they mainly check whether the condition is being fulfilled or not.

4.0.2 There are various control statements, like:

4.0.3 * if statement

4.0.4 * if-else statement

4.0.5 * if-elif-else staement

4.0.6 Also the control statement consist of the loops like :

4.0.7 * for loop

4.0.8 * while loop

4.0.9 if statement

It is a statement that is mainly used to check whether the condition is true.

[13]:
```
num = 10
if num==10:
  print("The number is 10")
```

```
The number is 10
```

4.0.10 if-else statement

It is a statement in which the if statement gets false, then the else statement gets executed.

[14]:
```
num = 1
if num ==2:
  print("The number is 2")
else:
  print("The number is not 2")
```

```
The number is not 2
```

4.0.11 if-elif-else statement

It is a statement in which the if statement gets false,then the elif statement having a different condition can get executed, and if the elif also gets false, then the else statement gets executed.

[15]:
```
num = 21
if num==10:
  print("The number is 10")
elif(num==21):
  print("The number is 21")
else:
  print("The number is neither 10 nor 21")
```

```
The number is 21
```

4.0.12 Loops

They are the iterative loops in which a condition is placed, and the loop runs until it hits the condition.

[17]:
```
fruits= ['apple','orange','banana','pineapple']
fruits
```

[17]:
```
['apple', 'orange', 'banana', 'pineapple']
```

While using the for loop for any data structure,. We must keep in mind that we need to take a temporary variable, as in the below code, the temporary variable is "fruit." . We can take any named variable (the most common are i and j).

[20]:
```
for fruit in fruits:
  print(fruit)
```

```
apple
orange
banana
pineapple
```

Here, loops are of two types, i.e., with and without range. The above example is without the range used in it. as the without range for loop gives the direct values from the list in the above example.

There is also a for loop with a range. range is nothing but the limit of how much time the loop will run.

[21]:
```
for i in range(5):
  print("Hello Learner")
```

```
Hello Learner
Hello Learner
Hello Learner
Hello Learner
Hello Learner
```

As here, I had given the range of 5, and the loop had run five times, printing the 'Hello Learner' five times.

4.0.13 while loop

It is the same as a loop, but it is running until the defined condition is met.

[22]:
```
num = 1
while num<=5:
  print(num)
  num +=1
```

```
1
2
3
4
5
```

As in the above example, when the while loop hits the condition of num <= 5, the loops gets breaked.

5 FUNCTIONS

5.0.1 Functions are the blocks of reusable code that perform a single task at a time.

[23]:
```
def add(num1,num2):
  return num1+num2
```

[24]:
```
add(2,3)
```

[24]: 5

Also, in the function, we are able to keep some default parameters so that whenever any parameter is missing, it can get a constant value.

[25]:
```
def add(num1=10,num2=12):
  return num1+num2
add()
```

[25]: 22

6 CLASSES-OBJECT

6.0.1 Classes

6.0.2 Classes in Python serve as blueprints for creating objects, bundling data and functionality together. Each class instance can have attributes for maintaining its state and methods for modifying that state. Objects are created from classes, allowing new instances of that type to be made. Attributes encapsulate the object's data, while methods define its behaviour, enabling manipulation of its state and the execution of specific actions.

[26]:
```
class Calculator:
    def add(self, x, y):
        return x + y
```

6.0.3 Objects

6.0.4 In Python, objects are instances of classes. When you create a class, you're essentially defining a blueprint or template for creating objects. These objects are specific instances of the class, with their own unique attributes and methods.

[28]:
```
#Here we have created a class
class Calculator:
    def add(self, x, y):
        return x + y
```

[30]:
```
#we are initializing the class to the object where the object will represent␣
 ↪the class
cal = Calculator()
print(cal.add(1,2))
```

```
3
```

Here I have covered the basics of Python in the Data Science series. As in week one, we covered the basics. In the coming week, we will see about the data analysis library's.

Thank You

[ ]:
```

```