Python in 4 sittings: Sitting 1

Introduction to Python

Summary of the first class

13th June, 2021

feel free to contact us;

arabindo@protonmail.com

kaustavbasu97@gmail.com

Basic Data Types

As usual we have **integer**s, **float**s and **string**s. Here, in Python you really don't need to worry about defining the data types explicitly. You can just keep using any particular variable with any data types at different instances of the program. Python interpreter will figure it out for out (lazyness is awesome! xD).

You can perform comparison operations with different data types (exception: strings). Example:

In [1]:
```
print(42==42.00)
print(42.00==00042.00)
print(42=='42') # '42' is a string and 42 is an integer. cool?
```

```
True
True
False
```

Although the result of the comparision operations in the first two lines have been evaluated as **true**, if you check their data types, they'll be different. For that you just need to type `type(variable)` or `type(data)`

In [2]:
```
var = 42
print(type(var))
print(type(42))
print(type(42.00))
print(type('42'))
```

```
<class 'int'>
<class 'int'>
<class 'float'>
<class 'str'>
```

```
`# This is formatted as code`
```

Now, here comes the **print** statement -

General syntax : `print("statements", variable1, variable2, ....)`

In [3]:
```
#examples,,,oho by the way a line starts with hash sign,
#it means it's a comment line
#python interpreter will skip those lines
a = 5
b = 6
print("we have number", a, 'and', b)
#one more thing single or double inverted comma will do the same job unles
#you're using inverted comma with in the statements.
#In that case you have to use thise comma alternatively.
```

```
we have number 5 and 6
```

Now let's move to the Control statements: Generic good old IF-ELSE

Let's say we have two numbers `a` and `b`, and we want to perform comparisions between them.

For taking input for a and b from the user, The `input()` function can be used. You can also add a prompt to it (although it's not mandatory), by `input("statements")` ** Remember, whenever you try store an user-input it gets stored as a string. Therefore, you must convert those inputs into number(or any other specific type). In the next example we have converted an input string to float. You can verify the input data type by using the `type(variable)` command, as we stated earlier, in line In[2]

In [4]:
```
a = float(input("enter a number: "))
b = float(input("enter another number: "))
if(a>b):
    print("a is grater than b")
elif(a<b): # it means else if, in some places it is not required,
            #and in some places it becomes convenient to have
    print("a is less than b")
else:
    print("a is equal to b")
```

```
enter a number: 10
enter another number: 15
a is less than b
```

Let's make it a bit prettier:

In [5]:
```
a = float(input("enter a number: "))
b = float(input("enter another number: "))
if(a>b):
    print(a, "is grater than", b)
elif(a<b):
    print(a, "is less than", b)
else:
    print(a, "is equal to", b)
```

```
enter a number: 15
enter another number: 15
15.0 is equal to 15.0
```

notice those are float point now, can you figure it out why so?

General Syntax:

```
if(logical expressions):
    instructions
    ...
    ...
elif(logical expressions): #elif and else blocks are optional
    instructions
    ...
    ...
else:
    instructions
    ...
    ...
```

indentations are important. While writing code you must focus on the indentation!

In other languages like **C** or **C++** or **Java**, indentation is an optional thing, i.e. even without indentations, the codes work properly. Do you know the reason? Well, we can use braces there to segmentify/group different regions of the script.

The purpose of indentation in python:

- **Segmentification/Grouping.**
- **Beautification**.

"Infinite" and "Finite" While Loop

Before that, let's look into boolean datatype. In Python, the values supported for the said are `True` and `False`. Python is a **case sensitive** language. So you have to be careful about it. The first letter is capitalized here 'T' and 'F'

We can have local operators too. like `and`, `or` and `not` just as you do in other languages. For example in **C** or **C++** the equivalent operators are **&&**, ||

Sometimes it is convenient to use infinite loop and to break the loop when certain conditions are met.

for example we will print 1 to 10 using an infinite loop. Whenever the varibale reach the value 10 the loop will `break` Let's try!

CAUTION: IF YOU RUN THIS CODE IT'LL START A NEVER ENDING (i.e. INFINITE) LOOP.

To STOP, CLICK ON THE SQUARE BOX AT THE TOP OF YOUR SCREEN

```
num = 0
while True:
    print(num)
    num = num + 1
```

In [6]:
```
num = 0
while True:
    num += 1 # shorthand for n=n+1
    print(num)
    if(num == 10) : break
    # you can put the break statement in the next indented line.
```

```
1
2
3
4
5
6
7
8
9
10
```

We can do the same job in a different way:

general syntax:

```
while(logical expression):
    statements
    ...
    ...
```

Now let's look into the familiar `for` loop

Syntax:

```
for var in range(start, end, step):
    statements
    ...
    ...
```

`start` and `step` arguments in the `range` are optional. By default those are set to be 0 and 1 respectively.

In [7]:
```
for i in range(2,17,3):
    print(i)
# Notice the output! It doesn't print 17.
# It's because the range function is evaluating "upto" the number 17
```

```
2
5
8
11
14
```

Let's talk about function. Perhaps the the second last building block of lerning a new language. Immediately after LEARNING the function you'll able to do the regular numerical program you used to write for your academic purpose.

Syntax:

```
def name(arguments):
    statements
    ...
    ...
    retuen the_resut
```

let's have a simple addition function.

In [8]:
```
def addition(a,b):
    c = a+b
    return c
sum = addition(6,7)
print(sum)
# or we can do
print("Let's try with ccompoisition of print and add function", addition(1
```

```
13
Let's try with ccompoisition of print and add function 21
```

We can have another interesting example. "Collatz" sequence.

The idea is; take any arbitrary integer. If it is even, divide it by 2 otherwise multiply the number by 3 and then add 1 to it. Repeat this process. It'll get terminated at n=1. But nobody knows why!

In [9]:
```
# if n is odd then n*3+1
# if n is even then n/2
def collatz(n):
    while (n!=1):
        if(n%2==0):
            n = n//2 # // means integer division
            print(n)
        else:
            n = 3*n+1
            print(n)
user_input = int(input("Enter a number="))
collatz(user_input)
```

```
Enter a number=30
15
46
23
70
35
106
53
160
80
40
20
10
5
16
8
4
2
```

So, we are at the very end of getting aquainted with a new programming language.

For any new language you want to learn, the basic steps are the same.

- Learn some words ......
- Learn the grammar ....
- Try to use them ....
- Repeat

.... And you're done!

Now, Let's talk about some inbuilt data structures

List

These are mutable, that is we can edit the elements of the list. Lists can contain heterogeneous data unlike C/C++/Fortran array. We denote list by a pair of brackets. Example;

In [10]:

```
a = [2,3,4,5]
b= ['a',2,3.45,['g','e']]
print(a)
print(b)
print(a[1])
print(b[2])
```

```
[2, 3, 4, 5]
['a', 2, 3.45, ['g', 'e']]
3
3.45
```

We can iterate over list by two different ways,

i. By index

ii. By the elements of the list itself

** NOTE: List index starts from 0

In [11]:

```
array = [2,['a','b'],'efg', 2.6]

print("Itterating over index")
#len(list_name ) return the length of the array
for i in range(len(array)):
    print(array[i])
print("------------")
print("Itterating over elements")
for elements in array:
    print(elements)
```

```
Itterating over index
2
['a', 'b']
efg
2.6
------------
Itterating over elements
2
['a', 'b']
efg
2.6
```

Let's have a 2-D list, that is a list of a list

We will go simple. Say, we have an idendity matrix

In [12]:

```
matrix = [[1,0,0],[0,1,0],[0,0,1]]
# Print the matrix
for element1 in matrix:
    print(element1)
print("----*****-----")
#access the each element and print them
for element1 in matrix:
    for element2 in element1:
        print(element2)
```

```
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
----*****-----
1
0
0
0
1
0
0
0
1
```

As we have said lists are mutable, let's have an example.

Also note that string and tuple are immutable.

You can convert one data type to another with smple built-in functions. like: `tuple(var)` , `list(var)` , `string(var)`

In [13]:
```
# We defined 'a' earlier, so we do not need to do it again,
# if you're doing this
# in ascript, you should be defining `a` otherwise you'll have an error
print(a)
a[2] = 'new'
print("edited a: ", a)
#convert 'a' in to tuple and assign it to a new variable, say b
b = tuple(a)
print('type of a:',type(a),'\n type of b: ', type(b))
# \n means it'll print it in a new line
```

```
[2, 3, 4, 5]
edited a:  [2, 3, 'new', 5]
type of a: <class 'list'>
 type of b:  <class 'tuple'>
```

In [14]:
```
#Let's try to change the element of the tuple
b[1] = 'x'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-1a5b0875d144> in <module>
      1 #Let's try to change the element of the tuple
----> 2 b[1] = 'x'

TypeError: 'tuple' object does not support item assignment
```

You see, it's not allowed. You can try to convert that tuple or list to a string and can try to do the same. String element can be accessed in the same manner.

Say if you have a string variable, `name='python'` then name[2] will return 'y'0. Try yourself!

Dictionary and all

Unlike string, tuple and list, dictionary are consists of key-value pairs. keys are string type data where as values could be anything - lists, string, tuples and even can be another dictionary. We will go easy. Let's say, you want to design a game and you want to keep track of gems, lifes and the user name of the player. btw, dictionaries are mutable objects!

In [15]:
```
#let us create a dictionary, `game`

game = {'gems':20,'life':3, 'user_name':'abc123'}
print(game)
print("data type of the game: ",type(game))
```

```
{'gems': 20, 'life': 3, 'user_name': 'abc123'}
data type of the game:  <class 'dict'>
```

In [17]:
```
# you can access the elements of the game in the following manner
print(game['user_name'])

# as dictionary is mutable you can change the value corresponding to a key
game['life'] -= 1 #equivalent to game['life'] = game['life'] - 1

print('life updated:',game['life'])
```
```
abc123
life updated: 1
```

In [18]:
```
#you can check a if a particular key exist in a dictionary or not
'token' in game
```
Out[18]:
```
False
```

In [19]:
```
'gems' in game
```
Out[19]:
```
True
```

In the session I was unable to show you `issmall` thins with strings. Actual command is `islower()` and `isupper()` ! My bad, sorry! :")

In [20]:
```
string = 'python'
if string.islower():
    string = string.upper()
print(string)
```
```
PYTHON
```

you can try the reverse

let's look into some other methods

`isalpha` - is the string is consists of anly alphabets

`isalnum` - is the string consists of alphanumeric type data

`isdecimal` - is the string consists only of decimals

In [21]:
```
a = '4Abc4543'
print(a.isdecimal())
print(a.isalnum())
print(a.isalpha())
```
```
False
True
False
```

The very last thing we did, we imported pyperclip

What it does? It is helpful to access your clip board

Before running the block below, copy some text from anywhere you like

In [22]:
```
import pyperclip
test = pyperclip.paste()
print(test)
```

```
I copied this text earlier.
```

That's it for today!!

Practise this elementary stuffs. We will upload the problem set soon, where you need to show some of your creativity to solve those problems. Remember, writing a code is just the 5%; the rest is bug fixing where the learning starts. Before start writing a code, try to write down the idea on paper, how you'll solve the problem. If you do that, you've already done with the 50% of your task. Also, we will introduce you to the `random` library in the problem set. Don't worry; we will always be there to solve your confusion and to make you comfortable. You're just an email away!

P.S. We'll try to make the next session less messy. ROFL!