Notebook_3_Basics_of_Python_19Aug2025

August 19, 2025

1 Basics on python and coding

1. Python is an interpreted, general-purpose programming language and supports procedural, functional and object-oriented paradigms.
2. It's core philosophy is that the code is beautiful, explicit, simple, and readable.
3. To promote this core philosophy, as users you need to keep certain things in mind. Your code should be simple, readable and well-documented.
4. Some key elements with regards to this course. It is important that your code works in the simplest possible manner. You can use fancy optimization and clever tricks later. Always, use a function whenever possible while writing a code, so that it can be reused.
5. Jupyter notebook (nb) is an interactive computing platform that spun out of IPython (Interactive Python).

1.1 1. Bash commands on Jupyter nb

You can write most of bash commands in Jupyter nb and execute them. For some functions, you may need to append the command with an **! sign**.

But note that commands that require user input such as **read** or **cat > file.txt**, may not be straightforward. So, let us keep those only for the terminal.

If you are writing a bash script, then use the header **%%bash** before writing the script as shown in the second example.

Bash on Jupyter is not important for examination purposes

[4]:
```
!pwd
!cat ~/Dropbox/PH4343_Autumn2023-24/Work_home/Aug4/quiz2.txt # you can create a␣
↪file elsewhere and open it here
!ls
```

```
/Users/hsdhar/Dropbox/PH4343_Autumn2023-24/Lectures/Notebooks
Aditi 25
Bharat 20
Chandru 10
Notebook_3_Basics_of_Python_18Aug2022.ipynb
```

[5]:
```
%%bash
vartext='Physics'
varnum=2022
greet="This is the $vartext batch of $varnum"
echo $greet
echo ''
a=2
b=3
c=$((a+b))
echo $c
```

```
This is the Physics batch of 2022

5
```

1.2 2. Numbers and string in Python - Dynamical type

As mentioned Python is a dynamically typed language. This implies that the type of the variables (integer, string etc.) do not need to be declared at the start, but are dynamically assigned while the code is interpreted.

[1]:
```
x = 14
print(x)  # Print the variable x, which we have not declared yet
type(x)   # The interpreter has dynamically assigned it the type integer (int)
```

```
14
```

[1]:
```
int
```

[7]:
```
name = 'Shahrukh Khan'
print(name)
type(name) # The interpreter has assigned it the type string (str)
```

```
Shahrukh Khan
```

[7]:
```
str
```

[8]:
```
name = 'Shahrukh Khan'
print(name)
print(type(name)) # The interpreter has assigned it the type string (str)

name = 14
print(name)
print(type(name)) # The interpreter has now dynamically changed it to the type␣
  ↪integer (int)
```

```
Shahrukh Khan
<class 'str'>
14
<class 'int'>
```

[9]:
```python
text = '''Hello, world!
How is life?
When will this class end!''' # You can define a multiline text as a string

print(text)
```

```
Hello, world!
How is life?
When will this class end!
```

[9]:
```python
num = input("Please enter a number:")
print(num)
print(type(num))
```

```
Please enter a number: 5

5
<class 'str'>
```

[11]:
```python
num = int(num)
print(num)
print(type(num))
num = float(num)
print(num)
print(type(num))
```

```
5
<class 'int'>
5.0
<class 'float'>
```

1.3 3. Integers, floats and complex number

A number in python can be stored either as an integer, a floating point (decimals) or a complex number

[10]:
```python
x = 100045666
print(x)
type(x)
```

```
100045666
```

[10]:
```
int
```

[11]:
```python
y = 25.3677
print(y)
type(y)    # The interpreter has assigned it the type floating point (float) --␣
 ↪in python all floats are double precision
```

```
25.3677
```

```
[11]: float
```

```
[12]: x = 1.000456668787787098890    # Only 15 places are kept after the decimal with␣
      ↪the 16th digit after rounded off
      print(x)
      type(x)
```

```
1.0004566687877872
```

```
[12]: float
```

```
[13]: x = 100045666
      y = 25.3677
      z = x+y
      print(z)
      type(z)   # The interpreter has now dynamically changed it to the type float
```

```
100045691.3677
```

```
[13]: float
```

Python float values are represented as 64-bit double-precision values. The maximum value any floating-point number can be is approx 1.8×10^{308}. Any number equal to or greater than this will be indicated by the string **inf** in Python.

```
[2]: print (1.79e308)
     print (1.8e308)
```

```
1.79e+308
inf
```

```
[3]: x = 2 + 3j # Defining complex numbers
     print(x)
     type (x)  # The interpreter has dynamically assigned it the type complex number␣
     ↪(complex)
```

```
(2+3j)
```

```
[3]: complex
```

```
[4]: print(x.real)  # use or print real part
     print(x.imag)  # use or print imaginary part
```

```
2.0
3.0
```

1.4 4. Tuples and lists

Tuples and list are used to store multiple items in a variable. While tuples are more restrictive, list allows for dynamic update of its elements.

1.4.1 Tuples

```
[5]: pet = ('dogs','cat') # This is a tuple
```

```
[6]: print(pet[0])    # You can access the elements in the tuple starting from index 0
     print(pet[1])
```

```
dogs
cat
```

```
[7]: pet[0] = 'fish' # You cannot change elements already assigned during definition
     pet.append('fish') # You cannot add or remove anything from tuple
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 pet[0] = 'fish' # You cannot change elements already assigned during␣
 ↪definition
      2 pet.append('fish')

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

1.4.2 Lists

```
[19]: pet = ['dogs','cat'] # This is a list
      print(pet[0])    # You can access the elements in the list starting from index 0
      print(pet[1])
```

```
dogs
cat
```

```
[20]: pet[0] = 'fish' # You can change elements already assigned during definition
      pet.append('fish') # You can add anything in a list
      print(pet)
```

```
['fish', 'cat', 'fish']
```

```
[26]: var = [3,4,'cat','fish'] # You can store different types in a list
      print (var)
```

```
[3, 4, 'cat', 'fish']
```

```
[27]: var.append('dog') # You can add or remove anything from tuple
      print(var)
      var.remove('cat')
      print(var)
```

```
[3, 4, 'cat', 'fish', 'dog']
[3, 4, 'fish', 'dog']
```

Accessing elements in a tuple and list

[30]:
```
number_list1 = list(range(20)) # range is used to create a list of int upto 20␣
  ↪(not included) starting from 0
number_list2 = tuple(range(10,20)) # range command is used to create a tuple of␣
  ↪int upto 20 starting from 10
```

[31]:
```
print(number_list1)
print(number_list2)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
(10, 11, 12, 13, 14, 15, 16, 17, 18, 19)
```

[32]:
```
print(number_list1[4:8])  # accessing elements in a list from 4th upto 8th␣
  ↪(exluding 15th)
print(number_list2[4:8])  # accessing elements in a tuple from 4th upto 8th␣
  ↪(exluding 15th)
```

```
[4, 5, 6, 7]
(14, 15, 16, 17)
```

[33]:
```
print(number_list1[-5])  # accessing the 5th element in reverse order
print(number_list2[-5])
```

```
15
15
```

[34]:
```
print(number_list1[:-5])  # accessing elements excluding last 5 (or upto the␣
  ↪last 5)
print(number_list2[:-5])
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
(10, 11, 12, 13, 14)
```

[35]:
```
print(2*number_list1[:-5])  # multiplying a integer n by a list or tuple,␣
  ↪creates n copies of the list or tuple
print(3*number_list2[:-5])
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14]
(10, 11, 12, 13, 14, 10, 11, 12, 13, 14, 10, 11, 12, 13, 14)
```

[36]:
```
print(number_list1+number_list1)  # adding two lists is the same as multiplying␣
  ↪by 2
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 0, 1, 2,
3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

[41]:
```
print(number_list2+number_list2)  # adding two tuples is the same as␣
  ↪multiplying by 2
```

```
(10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19)
```

[37]:
```
print(number_list1+number_list2)  # cannot add a list and a tuple
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[37], line 1
----> 1 print(number_list1+number_list2)

TypeError: can only concatenate list (not "tuple") to list
```

Two-dimensional lists and tuples

One can create a list of lists or a tuple of tuples. In fact, one can create a list of tuples and vice-versa, as long as specific rules for tuples and lists are adhered to.

[38]:
```
twoD_tuple = ((1,2),(3,4))  # tuple of tuples
twoD_list = [[1,2],[3,4]]   # list of lists
twoD_mix0 = ([1,2],[3,4])   # tuple of lists
twoD_mix1 = [(1,2),(3,4)]   # list of tuples
```

[39]:
```
print(type(twoD_tuple)) # is a tuple
print(type(twoD_list))  # is a list
print(type(twoD_mix0))  # is a tuple
print(type(twoD_mix1))  # is a list
```

```
<class 'tuple'>
<class 'list'>
<class 'tuple'>
<class 'list'>
```

[40]:
```
print(twoD_tuple)
print(twoD_list)
print(twoD_mix0)
print(twoD_mix1)
```

```
((1, 2), (3, 4))
[[1, 2], [3, 4]]
([1, 2], [3, 4])
[(1, 2), (3, 4)]
```

[33]:
```
print(twoD_tuple + twoD_mix0) # adding two tuples creates a bigger tuple
print(2*twoD_tuple)  # scalar multiplication of a tuple
```

```
((1, 2), (3, 4), [1, 2], [3, 4])
((1, 2), (3, 4), (1, 2), (3, 4))
```

[19]:
```
print(twoD_tuple[1][0]) # Accessing the 0th element in 1st tuple of␣
 ↪((1,2),(3,4))
```

```
print(twoD_list[0][1])
print(twoD_tuple[1][1])
print(twoD_list[0][0])  # Accessing the 0th element in 1st list of ((1,2),(3,4))
```

```
3
2
4
1
```

[43]:
```
# print(twoD_list[1][0])
twoD_list[1][0] = 35     # You can change the element of a list inside a list
print(twoD_list[1][0])
twoD_mix0[1][0] = 35    # You can change the element of a list inside a tuple
print(twoD_mix0[1][0])
twoD_tuple[1][0] = 35    # You cannot change the element of a tuple inside a␣
 ↪tuple
twoD_mix1[1][0] = 35    # You cannot change the element of a tuple inside a list
```

```
35
35
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [43], in <cell line: 6>()
      4 twoD_mix0[1][0] = 35    # You can change the element of a list inside a␣
 ↪tuple
      5 print(twoD_mix0[1][0])
----> 6 twoD_tuple[1][0] = 35    # You cannot change the element of a tuple␣
 ↪inside a tuple
      7 twoD_mix1[1][0] = 35

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

1.5 5. Tasks for today

Solve the following problems.

0. Create a list and tuple of the name and roll number of 5 students in your class.
1. Consider the 3 complex numbers: 5 + 4i, 3 + 3i, 7i. Create an array or matrix, where the real part is in first column and imaginary in second column. What is the size of the matrix?
2. Write a code that allows a user to define a random 2 x 2 matrix.
3. Write a code to add two matrices.
4. Write down a code to multiply two matrices.
5. Write a code that inputs a 3 x 3 matrix and calculates its trace and determinant.