exercise_01_intro-to-python_tutorial

November 15, 2023

1 Tutorial 1: Introduction to Python and Jupyter Lab

1.1 Install Jupyter Lab using Anaconda

1) Install Anaconda (or miniconda or mamba if you know what you are doing)
2) Install and run JupyterLab

Alternatively you can upload these tutorials to Google Colab (no installation required).

1.2 Variables and data types

Execute the cell with `Shift+Enter` or `Ctrl+Enter`:

Click on the left of a cell to get into the "blue" mode then you can press: * a -> new cell above the current cell * b -> new cell below the current cell * x -> delete the current cell

Click within a cell to get into the "green" mode (edit mode). Now you can write code here.

[1]:

```
# print statements give out information
print ("Hello")
print ("World")
```

```
Hello
World
```

[2]:

```
x = "Hello!"
print(x)

y = "Hello"
x = " World"
print (y + x)

x = 5
y = 2
print (x + y)
```

```
Hello!
Hello World
7
```

[3]:

```
x
```

[3]: 5

[4]:

```
print (4+5)
print (x == 10)
```

```
9
False
```

[6]:

```
x = "Hello"
y = "World"
print (x + y)
```

```
HelloWorld
```

[14]:

```
#Python is dynamically typed, types can change:
x = "hello"
print (x)
x = 5
print (x)
```

```
hello
5
```

[15]:

```
#main data types:
x = 5
print(type(x))

x = 5.0
print(type(x))

x = "5"
print(type(x))

x = True
print(type(x))
```

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

[7]:

```
# Python is strongly typed
x = "5"
y = 2
x+y
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-bca780087fd5> in <module>
```

```
      2 x = "5"
      3 y = 2
----> 4 x+y

TypeError: can only concatenate str (not "int") to str
```

[ ]:
```
type(x)
```

[ ]:
```
x = "5"
y = "4"
int(x)+int(y)
```

1.3 Data structures

1.3.1 Lists

[ ]:
```
#creating an empty list
l = []
```

[ ]:
```
# adding elements to the list
l.append (2)
l.append (5)
l.append (10)
l
```

[ ]:
```
l[1]
```

[ ]:
```
# accessing an element
l[-1]
```

[ ]:
```
l [:-1]
```

[ ]:
```
#setting an element
l[2] = 7.3
l
```

[ ]:
```
#the length of a list:
len (l)
```

[ ]:
```
# We can also directly specify lists:
# a list with integer objects
list_2 = [3,5,7,9]

# a list of strings
list_3 =␣
 ↪['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
```

```python
# a list of mixed types
list_4 = [2, 5, "Elephant", 6, 8.2, True]
```

[ ]:
```python
# we can also do lists of lists!
lol = [[1,2,3], ["a","b","c"], [1.2,2.3,4.5,6.7,8.9]]
```

[ ]:
```python
#the length of a list:
len (lol)
```

1.3.2 Tuples

Tuples are just the same as lists, but are *immutable*.

[ ]:
```python
t = (1,2,3)
```

[ ]:
```python
t[1]
```

[ ]:
```python
t[1] = 5
```

1.3.3 Sets

Sets are similar collections, but have no order and can contain each element only once

[ ]:
```python
s = set()
s.add(50)
s.add(20)
s.add(10)
s.add(20)
s
```

[ ]:
```python

```

[ ]:
```python
l = [2,3,4,5,6,2,3,4,5,2]
```

[ ]:
```python
s = set()
for x in l:
    s.add(x)
s
```

[ ]:
```python
list (set(l))
```

1.3.4 Dictionaries

Python's built-in mapping type. They map keys, which can be any immutable(unchanchable) type, to values, which can be any type

[ ]:
```python
d = dict()
d = {}
```

```
[ ]: presidents_inauguration = {}
     presidents_inauguration ['Trump'] = 2017
     presidents_inauguration['Obama'] = 2009
     presidents_inauguration['Bush'] = 2001
     print(presidents_inauguration)
```

```
[ ]: # or, shorter:
     presidents_inauguration = {'Trump': 2017,
                                'Obama': 2009,
                                'Bush': 2001}
```

```
[ ]: presidents_inauguration ["Trump"]
```

```
[ ]: print (presidents_inauguration.keys())
     print (presidents_inauguration.values())
```

```
[ ]: len (presidents_inauguration)
```

1.4 Control statements

control flow in Python noticeable does not use ANY (,),[,],{,},... Instead *indendation* determines what belongs to block of commands

1.4.1 If-elif-else

```
[ ]: x = 10
     if x > 5:
         print ('This is a big number!')
```

```
[ ]: #Note the difference
     x = 0
     y = 0
     if x > 5:
         x = x + 1
         y = y + 1
     print (y)

     x = 0
     y = 0
     if x > 5:
         x = x + 1
     y = y + 1
     print (y)
```

```
[ ]:
```

```
[ ]: x = 15
     if x > 20:
```

```
    print ('This is a very big number!')
elif x > 10:
    print ('This is a big number!')
else:
    print ('this is a small number!')
```

[ ]:

```
x = 10
command = 'increment'
if command =='increment':
    x = x + 1
print (x)
```

1.4.2 Loops

[ ]:

```
first_names = ['John', 'Paul', 'George', 'Ringo']
for name in first_names:
    print("Hello " + name + "!")
```

Contrary to many other programming languages there is no built-in for… counting loop. However, you can use the range function:

[ ]:

```
list (range(10,20,3))
```

[ ]:

```
for i in range (10):
    print (i)
```

[ ]:

```
# enumerate is a useful convenience functions:
for index, name in enumerate (first_names):
    print("Name "+ str(index) + ": " + name)
```

[ ]:

```
# We can loop over any iterable, e.g., also on strings
x = "example"
for letter in x:
    print (letter)
```

[ ]:

```
# As a simple example, lets create a dictionary, which maps each string in a␣
 ↪list to its name:
name_lengths = {}
for name in first_names:
    name_lengths [name] = len (name)
name_lengths
```

[ ]:

```
# while loops functions very similar to many popular languages:
x = 1
while True:
    x = x * 2
    if x >= 100:
        break
    print(x)
```

1.5 Functions

Defining your own functions is easy:

[ ]:
```
l = [1,2,3,4,5324,2,5,2,3,65,2]
len(l)
```

[ ]:
```
def print_all_names(names):
    for x in names:
        print (x)
```

[ ]:
```
print_all_names(l)
```

[ ]:
```
def increment_function(x):
    x = x + 1
    return x
```

[ ]:
```
increment_function(5)
```

[8]:
```
# You can call a function using its parameters names
def my_division (nominator,denominator):
    return nominator / denominator

print (my_division(12,4))
# you can call function parameters by name!
print (my_division(denominator=4, nominator=16))
```

```
3.0
4.0
```

[9]:
```
# You can also specify default parameters for a function
def my_division (nominator,denominator = 2):
    return nominator / denominator

print (my_division(12,3))
```

```
4.0
```

[10]:
```
# n
y = 5
x = 2
def increment_value (x):
    y = x + 1
    return y
increment_value(3)
y
```

```
[10]: 5
```

1.6 Imports

Python has a lot of built-in packages you can use, or you can download and install more packages from the internet. Using such packages is easy:

```
[ ]: import statistics

     statistics.mean([3,5,7,9])
```

```
[ ]: # you can also import just single functions from a package
     from math import log
     log (2.71 * 2.72)
```

1.7 List Comprehension

```
[ ]: my_list = [2,6,5,4,66,9,100,55,4,6,4,2]

     l2 = [2*x for x in my_list]
     l2
```

```
[ ]: l3 = []
     for x in my_list:
         l3.append(2*x)
     l3
```

```
[ ]: my_list = [2,6,5,4,66,9,100,55,4,6,4,2]

     new_list = [len(str(x)) for x in my_list if x > 20]
     new_list
```

```
[ ]: nl = []
     for x in my_list:
         if x > 20:
             nl.append(x*2)
     nl
```

1.8 numpy

```
[10]: import numpy as np

      x = np.array([1,2,3])
      x * 4
```

```
[10]: array([ 4,  8, 12])
```

```
[11]: m = np.array([[1,2,3],[4,5,6],[7,8,9]])
      m
```

```
[11]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])
```

```
[12]: m * 2
```

```
[12]: array([[ 2,  4,  6],
             [ 8, 10, 12],
             [14, 16, 18]])
```

```
[13]: m.dot(x)
```

```
[13]: array([14, 32, 50])
```

```
[14]: m[1,2]
```

```
[14]: 6
```