Lec13-Getting-started-with-iPython-Notebook

August 23, 2021

1 Getting started with Python

1.1 THIS IS A QUICK START TO PYTHON AND JUPYTER NOTEBOOK

1.1.1 (in the next chapter we go in more depth with some important comands and variables that appear here)

The best way to practice is to run all cells by yourself, but previously reset all the outputs. To reset, click on Kernel tab at the top and choose Restart and Clear Output. Let's not break the tradition! Let's start with "Hello, World!"

[2]:
```
print('Hello World!')
```

```
Hello World!
```

1.2 Create some variables in Python

[3]:
```
i = 4  # int
```

[5]:
```
# show the type of the variable i
type(i)
```

[5]:
```
int
```

[6]:
```
f = 4.1   # float (double precision, floating point numeric variable)
type(f)
```

[6]:
```
float
```

[7]:
```
z = 2 + 3j  # complex (j is i=(0,1) in R^2 representation of C)
            # there must be no space between 3 and j
print(z)
type(z)
```

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

[7]:
```
complex
```

[8]:
```
b = True   # bool (boolean i.e. logical variable)
print(b)
```

```
type(b)
```

```
True
```

[8]:
```
bool
```

[9]:
```
s = "This is a string!"
print(s)
type(s)
```

```
This is a string!
```

[9]:
```
str
```

1.3 Commenting

As you may have noticed, to tell Python to ignore certain content, such as your notes or a test code line, i.e. to treat a certain text as a comment, you start the comment with `#`. Everything after symbol `#` is ignored. You can also put a comment after a code line. For easier reading, you can often see some people commenting by starting with `##` (and Python woould treat the 2nd symbol `#` as the first character of the comment).

To have a comment in multiple lines, apart from starting with `#` in each line, you can also start and end the entire multiple line comment with `"""`. Unlike `#`, you should not have more than 3 characters `"`.

[8]:
```
## this is a comment line
# and this is the second line
## and the third one
bla = 7
bla
```

[8]:
```
7
```

[9]:
```
"""
this is another way
of commenting
in multiple lines
"""
bla = 7
bla
```

[9]:
```
7
```

1.4 Getting Info and Help

To look for basic info about a variable, type the question mark ? before or after the variable name

[10]:
```
## s was created above
?s
```

To look for basic info on a given command, type help(your_command)

[11]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

1.5 Key Shortcuts

By pressing Esc + H you get help on key shortcuts in both **Command Mode** and Edit Mode (for Edit Mode, scroll down in the window). This is also done by clicking on Help tab at the top of the notebook.

1.6 Advanced Variable Types

[10]:
```
# list (can contain components of different types)
li = [3, 1.0 , 8/2, "bla", 3+2j]

print(li)
type(li)
```

```
[3, 1.0, 4.0, 'bla', (3+2j)]
```

[10]:
```
list
```

[12]:
```
# dictionary  (in 'foo':1  foo is called key, and 1 is its value)
di = {'foo':1, 'bar':2.3, 's':'my first dictionary'}
di
```

[12]:
```
{'foo': 1, 'bar': 2.3, 's': 'my first dictionary'}
```

[13]:
```
di['bar']  #element of a dictionary
```

[13]:
```
2.3
```

`try-except` code chunks enable you to try certain command or set of commands, given in the `try` section. If it doesn't compile, do the alternative code chunk, given in the `except` section. There is a third (optional) section, `finally`, which executes code chunk regarless of `try` and/or `except` part.

[14]:
```
## the try block will generate an exepction, as di[1] is not allowed.
try:
    di[1]
except:
    print("Oops! You can't use indexes for calling a dictionary component; "\
          "think of keys as dictionary's indexes!")
finally:
    print("Anyhow, keys are dictionary's indices")
    print(di['bar'])
```

```
Oops! You can't use indexes for calling a dictionary component; think of keys as
dictionary's indexes!
Anyhow, keys are dictionary's indices
2.3
```

[16]:
```
# touple
tp = (-2, "bla", 7.3)
print(tp)
type(tp)
```

```
(-2, 'bla', 7.3)
```

[16]:
```
tuple
```

[17]:
```
# can also define tuple w/o parentheses
# unless the expression is too complicated

tp = -2, "bla", 7.3

print(tp)
type(tp)
```

```
(-2, 'bla', 7.3)
```

[17]:
```
tuple
```

[16]:
```
# range
ra = range(5)
print(ra)
print(len(ra))  ## length of the variable ra
type(ra)
```

```
range(0, 5)
5
```

[16]:
```
range
```

[17]:
```
# Python's null type
n = None
type(n)
```

[17]:
```
NoneType
```

1.6.1 To recapitulate:

Basic Types

`int`: integer

`float`: real number (double precision, floating point number)

`complex`: complex

`boolean`: logical

`str`: string (character)

Advanced Types `list`: a list (can be thought of what is "array" in some other languages)

`tuple`: n-tuple, an ordered set of length $n \in \{0, 1, 2, 3, \dots\}$

`range`: range variable, for iterative loops (i.e. for sequencing)

NOTES

- `int`, `float` and `complex` are called numeric types
- `list`, `tuple` and `range` are called sequence types; they are iterable, you can run a loop through them - see below
- There are many other types of variables/data; some of them built in, but many others are defined within particular modules/libraries/packages (such as `ndarray` from `numpy` module, or `Series` and `DataFrame` from `pandas` module)

1.7 Advanced printing

[18]:
```
i = 4; f = 4.1

print("Our float value is %s. Our int value is %s." % (f,i))  #Python is pretty␣
↪good with strings
```

```
Our float value is 4.1. Our int value is 4.
```

1.8 Getting and Changing Working Directory

[19]:
```
## import (i.e. load) module (i.e. package) os;
## this is equivalent to R's function library()
import os

## print current working directory; note that you need to specify
```

```
## the parent package of the child function getcwd()
mypath = os.getcwd()

mypath
```

[19]: '/Users/slan/Teaching/ASU/DAT301/Lectures'

[4]:
```
[type(mypath), len(mypath)]
```

[4]: [str, 40]

[5]:
```
## to get without double backslashes, use print command (which is "as is")
print(mypath)
```

```
/Users/slan/Teaching/ASU/DAT301/Lectures
```

[2]:
```
## change working directory using os function chdir
## Specify the destination path in the argument. It can be absolute or relative.
 ↪ Use '../' to move up.
## You can change the current directory in the same way as the UNIX cd command.

os.chdir('../')

print(os.getcwd())     ## didn't have to use "print", but just emphasizing
```

```
/Users/slan/Teaching/ASU/DAT301
```

[3]:
```
os.chdir('./')

print(os.getcwd())
```

```
/Users/slan/Teaching/ASU/DAT301
```

1.9 Conditional statements in Python

[26]:
```
i = 2; f = 4.3  ## you can write multiple statements in a single line,␣
 ↪separating them by ;

if i == 1 and f > 4:
    print("The value of i is 1 and f is greater than 4.")
elif i > 4 or f > 4:
    print("i or f are both greater than 4.")
else:
    print ("both i and f are less than or equal to 4")
```

```
i or f are both greater than 4.
```

[27]:
```
#you could also do this way (it's complicated, so, not recommended, at least␣
 ↪not for long statements)
```

```
print("The value of i is 1 and f is greater than 4.") if (i == 1 and f > 4)␣
↪else print("i or f are both greater than 4.") if i > 4 or f > 4 else print␣
↪("both i and f are less than or equal to 4")
```

```
i or f are both greater than 4.
```

1.10 for and while loops

[28]:
```
li = [3, 1.0 , 8/2, "bla", 3+2j]
print(li)
```

```
[3, 1.0, 4.0, 'bla', (3+2j)]
```

[29]:
```
for elem in li:
    print(elem)
```

```
3
1.0
4.0
bla
(3+2j)
```

Note that in Python we don't use {} or other markers to indicate the part of the loop that gets iterated. Instead, we just indent and align each of the iterated statements with spaces or tabs. (You can use as many as you want, as long as the lines are aligned.)

[30]:
```
counter = 6
while counter < 10:
    print(counter)
    counter += 1  ## I personally don't like +=, but this is certainly used by␣
↪many programers.
                  ## Instead, I prefer (that's just me) to write:  counter =␣
↪counter + 1
```

```
6
7
8
9
```

2 Creating functions in Python

Again, we don't use {}, but just indent the lines that are part of the function.

[31]:
```
## indentation is important!

def add2(x):
    y = x + 2
```

```
    return y
```

[32]:
```
add2(5)
```

[32]: 7

We can also define simple functions using reserved word `lambda` (so called lambda expressions or anonymous functions):

[33]:
```
square = lambda x: x*x

square(8)
```

[33]: 64

[1]:
```
## loading pandas module/package with alias pd
import pandas as pd
```

[3]:
```
#set/change directory to be the one from which we need some file for the next␣
 ↪cell.
os.chdir("./Lectures")
%pwd  ##print working directory; could do it with os.getcwd() as well
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-b6e4031bcee8> in <module>
      1 #set/change directory to be the one from which we need some file for th␣
 ↪next cell.
----> 2 os.chdir("./Lectures")
      3 get_ipython().run_line_magic('pwd', ' ##print working directory; could␣
 ↪do it with os.getcwd() as well')

NameError: name 'os' is not defined
```

[4]:
```
## reading the file, which is located in the current working directory
cd = pd.read_csv("WAC2019-100m-women.csv")
cd   ## note the html format of the presentation of the table
```

[4]:

	Rank	Lane	Name
0	1st place, gold medalist(s)	6	Shelly-Ann Fraser-Pryce
1	2nd place, silver medalist(s)	7	Dina Asher-Smith
2	3rd place, bronze medalist(s)	4	Marie-Josée Ta Lou
3	4	5	Elaine Thompson
4	5	8	Murielle Ahouré
5	6	9	Jonielle Smith
6	7	3	Teahna Daniels
7	NaN	2	Dafne Schippers

```
                  Nationality   Time Notes
0               Jamaica (JAM)  10.71    WL
1   Great Britain & N.I. (GBR)  10.83    NR
2           Ivory Coast (CIV)   10.9   NaN
3               Jamaica (JAM)  10.93   NaN
4           Ivory Coast (CIV)  11.02    SB
5               Jamaica (JAM)  11.06   NaN
6         United States (USA)  11.19   NaN
7           Netherlands (NED)    DNS   NaN
```

[9]:
```
print(cd)  ## plain printout, not an html printout
```

```
                            Rank  Lane                    Name  \
0    1st place, gold medalist(s)     6  Shelly-Ann Fraser-Pryce
1  2nd place, silver medalist(s)     7        Dina Asher-Smith
2  3rd place, bronze medalist(s)     4      Marie-Josée Ta Lou
3                              4     5         Elaine Thompson
4                              5     8         Murielle Ahouré
5                              6     9          Jonielle Smith
6                              7     3          Teahna Daniels
7                            NaN     2         Dafne Schippers

                  Nationality   Time Notes
0               Jamaica (JAM)  10.71    WL
1   Great Britain & N.I. (GBR)  10.83    NR
2           Ivory Coast (CIV)   10.9   NaN
3               Jamaica (JAM)  10.93   NaN
4           Ivory Coast (CIV)  11.02    SB
5               Jamaica (JAM)  11.06   NaN
6         United States (USA)  11.19   NaN
7           Netherlands (NED)    DNS   NaN
```

[ ]: