Python exercises

Mac keyboard combinations

{}	: alt+(and alt+)
[]	: shift (maj)+alt+(and shift (maj)+alt+(
~	: alt+n

Exercise 1 :

1. open a terminal :
   a. top of the screen
   b. Aller / Terminal
2. In the terminal, experiment with unix commands (havre a look at the quick reference folder) :
   a. $>pwd : your current working directory
   b. $>ls
      $>ls –l (this is an l as in <l>ist)
      list my files and folders in local directory
   c. $>cd /
      change directory to root (/)
   d. $>cd ~
      go back to my login ("~" is your default folder)
   e. $>man cd
      to get help about a command, try the following 2 commands.
      If you have many pages use the space bar to jump from page to page. Type "q" to quit.
      $>man ls
      $>man pwd
   f. Another usefull command is "apropos". If you do not remember the name of a command apropos can help you to discover the name of the commands of interest:
      $>apropos folder
      $>apropos file
      $>apropos network
      $>apropos video
      $>apropos game
   g. under "~", the logging directory, create your project folder. Verify that you are in your login folder ("~") with "pwd", this command should give something with "????/depulpXX" at the end. Once in "~", create the folder with :
      $>mkdir pythonProject
   h. move to this folder :
      $>cd pythonProject
      the folder is created under "~" the logging directory. This is a relative move
   i. you can move around folders with cd :
   j. $>cd ..
      You go up one level relative to the folder where you are
      $>cd ./pythonProject
      "." : is the folder where you are. Try :
      $> ls .
      An equivalent command if you are in the "~" folder would be :
      cd ~/ pythonProject or cd pythonProject, these commands should bring you back to the project folder

      But normaly we will use little linux commands : cd, ls, pwd, mkdir, cp, more

Exercise 2 : Python in interactive mode

1. You should be in the terminal under your "pythonProject" folder
2. Python round trip
   a. Run the python interpreter :
      i. $>python
         after this command you should see « >>> » the triple ">" signs you know that you are in python
      ii. But as I told you we use python3. Verify the version of python. Read the version number in the upper left corner.

        iii. This is version 2.7, the wrong version!!!
   b. exit python 2.7, try exit or quit :
      $>quit
      python shows you the right syntax, you should enter "quit()", but an even shorter syntax exists :
      ctrl^D (combine control key : lower left corner of keyboard and "d" character)

   c. run version3 of the python interpreter :
      >python3
   d. again you should see the triple « > » symbols:
      >>>
   e. check the python version
3. We are now ready to use python version3 as a calculator
   a. Try the following numerical operations +, -, *, / (float division), // (integer division), % (remainder of the integer division), ** (power) and of course parenthesis : "(" and ")"
   b. >>>10*4
   c. Replace * in b), d) with the other operators and try more complex expressions like :
      >>>5.5*(2+3)-20

Exercise 3 : Python variables (slides)

1. We can define variables by giving them a legal name and assign a value to them.
   a. >>>temperature=100 #this is a comment in python
   b. >>>temperature
      ???
   c. print(temperature)
   d. >>>i=10
   e. >>>_t=10.5
   f. >>>myname='Wurtz Jean-Marie'
   g. >>>my_name='Wurtz Jean-Marie'
   h. >>>myName='Wurtz Jean-Marie'
   i. Uppercase and lowercase are different
      >>>A=1
      >>>a=10
      "A" and "a" are different variables check their value
   j. Multiple assignations are allowed :
      >>>a=b=7
      >>>x, y=10.5, 20.7
   k. Use these variables in more complex expressions with the operators from the previous exercise and the parenthesis.
   l. You have also shortcuts, for example :
      >>>a = a+1
      Can also be written
      >>a += 1
      This is true for all the numeric operators (-, /, // and %)
   m. What gives :
      >>>3*1**10
      59049 or 3
      operators have priorities: for example: +, - (left to right) < *, / (left to right)
      if you are unsure how to evaluate an expression with the operators use parenthesis.

Exercise 4 : Python predefined functions (slides)

1. Try the following predefined functions in python :
   a. >>>help(print)
   b. >>>help(quit)
   c. >>>type(i)
   d. >>>type(_t)
   e. >>>id(i)
   f. >>>id(_t)
   g. >>>dir() # gives you the default scope (the known entities : vairables, functions, classes, …)
      what do you see ?
   h. >>>dir(i)
      what you obtain here is the scope of an integer variable. These are all the functions defined for any

integer in python.

i. >>>dir(__builtins__)   # enter 2 underscore symbols (__) : start and end
   This command shows you all the predefined functions/entities in python, you know already: quit, print, id, type, dir, help.
   Others are to be discovered like: min, max, sum, abs , int, …
   With the following instruction you should see the same as with dir(i), do you have any idea why?
   >>>dir(int)
   If you have now clue try the following :
   >>>type(i)
   ??
   What is the relation between the variable “i” and the predefined function “int”?

<u>Exercise 5 : String variables in python (slides)</u>

1. Define the following variable :
   >>>msg= "hello"
   What gives “msg” alone?
   >>>msg
   ??
   and print(msg) ?
   >>>print(msg)
   ??
2. Use the predefined functions id() and type() with the variable “msg”
3. Try the following experiment
   a. >>>s=msg*3
      ???
   b. >>>s=msg + " world"
      ??
4. Convert numbers to strings :
   a. >>>s1=str(12)
   b. >>>s2=str(100.99)
   c. >>>dir()   # you should see __builtins__
      >>>dir(__builtins__)
      and search “str” in the list.
      “str” is the “mother” entity for all strings. “str” is equivalent for strings as is “int” for integers
   d. >>>dir(s1)
      >>>type(s1)
      >>>dir(str)
      will give all the functions available to strings. For example “format()” that we will discover in the next exercise.
      But remember that with dir(str) you can discover rapidly the functions available for strings like split, stratswith, join
   e. >>>help(str)
      Here you discover that “str” is a “class”. This “str” is the skeleton or framework from which all the strings are constructed. Indeed, the “str()” function, when used as in “str(12)”, is called a “constructor function” of this class. More to classes later.

<u>Exercise 6 : String formating and print (slides)</u>

1. Use dir() and help() and try the python online web documention : python.org (select the correct version >= 3.0)
2. To use the “format()” function that belongs to the “str” class, we need a string variable to call such function : stringVar.format() or stringConstant.format().
   >>>s='a = {}, b = {}'
   >>>s.format(10, 20)
   >>>' a = {}, b = {}'.format(10, 20)
3. Try the few example from my slides :
   a. >>>book="Think python"
   b. >>>author="Allen Downey"
   c. >>>print("{} is the author of {}".format(book, author))
   d. >>>s= "{} is the author of {}".format(book, author)
   e. >>>"{0} is the author of {1}".format(book, author)
   f. >>>"{1} is the author of {0}".format(book, author)

g. >>>"{a} is the author of {b}".format(b=book, a=author)
h. >>>"{b} is the author of {a}".format(b=book, a=author)
i. >>>"a number {:4d}".format(12)
j. Different numerical format :
```
        >>>'{:d}'.format(10)
        >>>'{:x}'.format(10)
        >>>'{:o}'.format(10)
        >>>'{:b}'.format(10)
```
k. Examples from the python web site
```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
>>> '{}, {}, {}'.format('a', 'b', 'c')          # python >= 3.1 only
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
>>> '{2}, {1}, {0}'.format(*'abc')              # unpacking argument sequence
>>> '{0}{1}{0}'.format('abra', 'cad')           # arguments' indices can be repeated
```
1. The “print” predefined function can take two special arguments : sep (separator) and end. Try the following code :
```
>>>for i in range(5):
...         print(i)
…
>>>for i in range(5):
...         print(i, end=":")
…
>>>for i in range(5):
...         print('a',i, end=":")
…
>>>for i in range(5):
...         print('a',i, end=":", sep='=')
…
```

Exercise 7 : Write my own function (slides)

1. A function is defined with the keyword “def” :
```
>>>def hello():
…       print('hello')
…
>>>hello()        # here you call your function
'hello'
>>>type(hello)
???
```
2. The previous function has no arguments (names between parentheses) and only one instructions
```
>>>def hello2(firstname, lastname):
…       s=   'hello {} {}'.format(firstname, lastname)
…       print(s)
…
```
not the the « : » at the end of the definition of the function. « : » that a block of instructions will followed more on that in the next exercise
3. Functions are defined to avoid repetitive code. For example to calculate the surface of a rectangle is writen in python as :
```
>>>length=100
>>>width=20
>>>surf = length * width
>>>surf or print(surf)                # to obtain the result
```
either instruction will give the result in the “interactive mode” but only print() will display the result in “scripting mode”. If you want to recalculate the surface with different values the same statements are typed in. To avoid to repeat the code functions are introduced.
```
>>>def surface(l, w):
…       surf = l * w
…       return surf
…
>>>surf1 = surface(length, width)
>>>surf2 = surface(55.5, 10.5)
```
4. If among the function parameters a values appears often, like for example width=5.0 for out surface example, the surface function can be defined as :
```
>>>def surface(l, w=5.0):
```

```
…        surf = l * w
…        return surf
…
>>>surf3 = surface(20)   # default value for "w" will be 5.0
```

You can also give the width value with this new definition:

```
>>>surf4 = surface(20, 15.5)
```

We stop here for functions for now.

Exercise 8 : Control flow : if statement (slides)

1. Check if a temperature is positive, negative or null :

```
>>>temperature=10.5
>>>if temperature > 0 :
…       print('temperature is positive')
…elif temperature < 0 :
…       print('temperature is negative')
…else :
…       print('temperature is null')
…
```

2. Transform this code into a function : tempLevel1(temp)
3. Transform "templevel1" so that it does not print anything but return one the values -1, 0, 1 if temperature is negative, null or positive

Exercise 9 : Control flow : while statement (slides)

1. Print the 10 first integer square numbers :

```
>>>i=0
>>>while i <=10 :
…       print('{:3d}:{:5d}'.format(i, i*i))
…       i +=1
…
```

2. Write a similar code to print the even numbers among the first 20 numbers. Remember the "%" operator may be useful (4%2 is equal to 0 and 3%2 is equal to 1). Write the code as a function and try various values.
3. Two important instruction for control flow are :
   a. break : stop processing and exit the actual instruction block

   ```
   >>>i=0
   >>>while i <=10 :
   …       print('{:3d}:{:5d}'.format(i, i*i))
   …       if i == 3 : break
   …       i +=1
   …
   ```

   b. continue : do not pursue processing but stay in instruction block and go to next step

   ```
   >>>i=0
   >>>while i <=10 :
   …       if i == 3 :
   …           i +=1
   …           continue
   …       print('{:3d}:{:5d}'.format(i, i*i))
   …       i +=1
   …
   ```

Exercise 10 : Control flow : for statement (slides)

1. Print the 10 first integer square numbers :

```
>>>for i in range(10) :
…       print('{:3d}:{:5d}'.format(i, i*i))   # watch out the quotes there is a difference between <'> and <'>
…
```

2. I introduced a new predefined function "range()". This function generates a bunch of numbers, in our case : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10 is excluded.
   The complete syntax would be : range(start, end, step).
   So range(0, 10, 1) is equivalent to range(10).
   Another notation would is possible range(start, end). So range(0, 10) is equivalent to range(10).
   Range is a predefined function that you can find with :

   ```
   >>>dir(__builtins__)
   ```

```
>>>help(range)              # or check the python 3.x documentation
```
3. You can try and generate another series of numbers like with :
```
>>>print(list(range(-10,+10)))      # do not forget the list
>>>print(list(range(1, 30, 3)))
```
4. Write a similar code as in 1) to print the uneven numbers among the first 20 numbers. Remember the "%" operator may be useful (4%2 is equal to 0 and 3%2 is equal to 1). Write the code as a function and try various values

Exercise 11 : Data structures : lists (slides)

1. Use dir() and help() and try the python online web documention : python.org (select the correct version >= 3.0)
2. Function that operate on lists :
```
>>>lst1 = [10, 100, 1000, 10000, 100, 10]   # define a list
>>>dir(lst1)                        # you should see a bunch of names : variables and functions
>>>type(lst1)
```
what is the type of lst1?
do a dir(type of list). What do you conclude? Remember strings and integers.
3. Length of a list
```
>>>len(lst1)                        # the length of the list
>>>lst1[2]                          #get the third element in the list
```
lists start at 0 !!!
4. Check that an element belongs to a lists
```
>>>100 in lst1              # checks if 100 is in the list
>>> 1 in lst1
```
"True" and "False" are two constants defined in python that you can use
5. Loop over a list as in the for-statement as in exercise 10.1.
```
>>>for elt in lst1:
…       print(elt)
…
```
6. Other function for lists
```
>>>lst1.count(10)                   # how many times does 10 appear in the list
>>>lst1.index(100)                  # at what position do I find 100
```
Notice the difference in calling "len()" and "count()" or "index()"
7. Introducing a second list
```
>>>lst2 = list(range(1,10))         # create a list based on the series generated by range
>>>lst2
????
```
8. Introducing a third list : the result of a concatenation
```
>>>lst3 = lst1 + lst2
>>>lst3
????
>>>lst2.append(50)
>>>lst2
????
```
Can you discover other functions defined for lists such as append(), count(), …
9. Suppress an element
```
>>>lst1
>>>del lst1[2]
>>>lst1
```
10. Be careful when you think you copy a list as in the following example :
```
>>>lst5 = lst1
>>>lst5         #is identical to lst1: is it a copy or the same list
>>lst5[2]=-1000
>>>lst5
>>>lst1         #what do you conclude
>>>id(lst1)
>>>id(lst5)
```
"id()" is another predefined function that reveals the unique identifier assigned to python entity.
11. Convert a string to a list of characters
```
>>>abcdList=list('abcd')
>>>abcdList
```

12. Convert a list of characters to string
    ```
    >>>str(abcdList)
    ```
    this is not the right way
    ```
    >>>":".join(abcdList)          # join the characters and between them add ":"
    >>>"".join(abcdList)           #join the characters and between them add nothing (empty string)
    ```

    We stop here for lists today.

Exercise 12 : Slicing in python : sequences, string and lists (slides)

1. Slice operation apply to data structures that are sequences : ordered entities. The sequences we have seen are strings (str) and lists (list). The notation is the same for list and string, except that strings cannot be modified (string are immutable).
   The slice operator has the following notation, similar to range, "start : end : step".
   If start is omitted the default value is 0.
   If end is omitted the default value is the end of the list.
   If step is omitted the default value is 1.
2. Let's start with a list (you can try the same operations on strings later on):
   ```
   >>>nb = list(range(15))
   >>>nb
   ```
3. Take an interval of a list
   ```
   >>>nb[3:11]
   >>>nb[3:11:2]
   >>>nb[::3]
   ```
4. Negative indexes are allowed :
   ```
   >>> nb[:-2:2]
   ```
5. Modify or add elements (only for lists)
   ```
   >>>nb[ 2 :2]=[1000]
   >>>nb
   >>>nb[5 :5]=['toto', 'tata']
   >>>nb
   ```
6. Suppress or replace elements (only for lists)
   ```
   >>>nb[5:7]=[]    # [ ] is an empty list
   >>>nb
   >>>nb[2:3]=[2000]
   >>>nb
   ```

Exercise 13 : Python in script mode (slides)

1. Grab the "helloyou.py" script under the folder you have downloaded and store this script in your project folder. Then run the following command :
   ```
   $>python helloyou.py
   ```

   This is the way a python script can be executed as command line
   Sometimes you will also see an additional directory "__pycache__" that contains the compiled files of your project with the "pyc" extension.

   But in the next exercises we will use "idle" a simple python IDE. Launch the following command:
   ```
   $>idle
   ```
   You have one windows with the python interpreter as before.
2. Open the "helloyou.py" : File/Open and select the the "helloyou.py" script.
3. A second window opens, and you can run the script under : Run/Run module.
   The script should work as before.
4. Explore "idle" menus (top menu bar : File, Edit, Shell, Debug, Options, Window and Help)
5. Switch now to the "hangman game" project (hangman_python-exercices-v4.pdf)