Python for biologists

Tutorial 1 - Basic Python syntax and objects

by Tobias Andermann

In this tutorial we will be using two packages: `re` and `numpy`. `re` is part of the standard python distribution, i.e. you don't have to separately install it first. In case of `numpy` you may have to install the package first. You do this via the bash command line using the python package installer `pip`:

```
pip install numpy
```

(It may actually work running this from your spyder window (depending on the spyder version) without having to go back to the bash command line.)

After installing numpy see if the installation worked by loading the package:

In [ ]:
```
import numpy
```

Strings

Define a string:

In [1]:
```
my_string = 'I FREAKING LOVE PROGRAMMING'
```

You can use the `print()` function to print any object to the screen, e.g.:

In [2]:
```
print(my_string)
```

```
I FREAKING LOVE PROGRAMMING
```

Create a string consisting of N repititions of a character, e.g. 50 repitions of the string we defined above:

In [3]:
```
my_string*50
```

Out[3]:
```
'I FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMI
NGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMM
INGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAM
MINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRA
MMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGR
AMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROG
RAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PRO
GRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PR
OGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE P
ROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE 
PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE
PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE
PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE
PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE
PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE
PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE
PROGRAMMINGI FREAKING LOVE PROGRAMMINGI FREAKING LOVE PROGRAMMING'
```

Define a string containing a variable:

In [4]:
```
string_with_variable = 'I used to say: %s' %my_string
print(string_with_variable)
```

```
I used to say: I FREAKING LOVE PROGRAMMING
```

In the command above we used `%s` as a spaceholder for a **s**tring that we define after the quotation marks. If we want to insert an **i**nteger instead, the spaceholder would be `%i`, while for a **f**loat it woudl be `%f`. If you are inserting multiple variables into a string, you need to list these in parantheses after the quotation marks. See the following example:

In [5]:
```
name = 'Olaf'
age = 30
time = 3.8
statement = "My name is %s, I'm %i years old and I can run 1000m in %f minutes." %
print(statement)
```

```
My name is Olaf, I'm 30 years old and I can run 1000m in 3.800000 minutes.
```

You see that the float is by default expressed with 6 decimal points accuracy. In the example above that is not necessarily how we want to express it, but instead we would prefer only one decimal point. You can format a float within a string using the notation `%.Nf`, where N is the number of decimal points, which would be `1` in our case:

In [6]:
```
statement = "My name is %s, I'm %i years old and I can run 1000m in %.1f minutes."
print(statement)
```

```
My name is Olaf, I'm 30 years old and I can run 1000m in 3.8 minutes.
```

Select part of a string, e.g. the 9th-15th letter. Python starts counting with 0 and the last element in the brackets is the upper boundary but is not within the selection (standard for all programming languages, except R).

In [7]:
```
statement[8:15]
```

Out[7]:
```
'is Olaf'
```

Select everything starting from the 6th character:

In [8]:
```
statement[5:]
```

Out[8]:
```
"me is Olaf, I'm 30 years old and I can run 1000m in 3.8 minutes."
```

Print only the last chracter of the string:

In [9]:
```
statement[-1]
```

Out[9]:
```
'.'
```

Print the last 8 characters of the string

In [10]:
```
statement[-8:]
```

Out[10]:
```
'minutes.'
```

The `.replace()` function is native to python and does not need any special package import. It can be applied to any string object in order to replace one character with another. If you want to replace all `T`s in a string (`my_string`) with `X`s, you can use this command like this:

In [11]:
```
statement.replace('e','X')
```

Out[11]:
```
"My namX is Olaf, I'm 30 yXars old and I can run 1000m in 3.8 minutXs."
```

The `replace()` function only takes a single character to be replaced. If you want to replace multiple characters at the same time or introduce regular expressions (e.g. replace all numbers in a string) you can use the `sub()` function of the `re` package (`import re`). We can use it e.g. to replace everything that's not a vowel or a space. The `^` at the beginning of the expression in parantheses tells the function to **replace everything except** the following characters:

In [12]:
```
import re
re.sub('[^AEIOUYaeiouy ]','N',statement)
```

Out[12]:
```
'Ny NaNe iN ONaNN INN NN yeaNN oNN aNN I NaN NuN NNNNN iN NNN NiNuNeNN'
```

Lists

Define a list:

In [13]:
```
my_list = ['this','is','a','list']
```

Use the `print()` function to print your list to the screen:

In [14]:
```
print(my_list)
```

```
['this', 'is', 'a', 'list']
```

Select a list element, e.g. the second element (same indexing logic as described in the string-section above):

In [15]:
```
my_list[1]
```

Out[15]:
```
'is'
```

Turn a string into a list by splitting it at a given character, e.g. at each whitespace (let's use the stirng we defined at the beginning of the tutorial called `my_string`):

In [16]:
```
new_list = my_string.split(' ')
new_list
```

Out[16]:
```
['I', 'FREAKING', 'LOVE', 'PROGRAMMING']
```

Replace the last list element:

In [17]:
```
new_list[-1] = 'PYTHON'
new_list
```

Out[17]:
```
['I', 'FREAKING', 'LOVE', 'PYTHON']
```

Add element to a list

In [18]:
```
new_list.append('BECAUSE')
new_list
```

Out[18]:
```
['I', 'FREAKING', 'LOVE', 'PYTHON', 'BECAUSE']
```

Concatenate two lists

In [19]:
```
another_list = ['IT','IS','SO','MUCH','FUN']
concatenated_list = new_list + another_list
concatenated_list
```

Out[19]:
```
['I', 'FREAKING', 'LOVE', 'PYTHON', 'BECAUSE', 'IT', 'IS', 'SO', 'MUCH', 'FUN']
```

Turn a list into a string:

In [20]:
```
new_string = ' '.join(concatenated_list)
new_string
```

Out[20]:
```
'I FREAKING LOVE PYTHON BECAUSE IT IS SO MUCH FUN'
```

Arrays

Arrays are similar to lists but are much handier when working with numbers (floats or integers). The numpy package offers many useful functions for arrays.

Generate a random float between 0 and 1:

In [21]:
```
import numpy as np
np.random.random()
```

Out[21]:
```
0.7308284361367227
```

Create an array with 20 random floats between 0 and 1 using numpy.

In [22]:
```
my_array = np.random.uniform(0,1,20)
print(my_array)
```

```
[0.49966913 0.88431121 0.99711678 0.09862122 0.09312464 0.02653386
 0.27142403 0.12915814 0.8940648  0.20563897 0.04690354 0.26584832
 0.35748078 0.2087932  0.93046033 0.08815069 0.63962635 0.34259445
 0.79423455 0.74712427]
```

Calculate the mean of the array:

In [23]:
```
np.mean(my_array)
```

Out[23]:
```
0.42604396311236326
```

Calculate standard deviation:

In [24]:
```
np.std(my_array)
```

Out[24]:
```
0.3301536088239003
```

Generate a random integer between 0 and 99 (Note: upper boundary is outside of sample):

In [25]:
```
np.random.randint(0,100)
```

Out[25]:
```
95
```

Generate 20 random integers between 0 and 99:

In [26]:
```
np.random.randint(0,100,20)
```

Out[26]:
```
array([ 3,  9, 37, 45, 93, 44,  5, 35, 41, 26, 61, 29, 92, 26, 64, 45, 70,
       91, 82, 51])
```

Task: Simulate 10, 100, 1000, 10000, 100000 dice throws (1-6) and calculate the mean value (= expected value). What do you observe with increasing sample sizes?

General tools

You can check the type of an object by typing:

In [27]:
```
type(my_list)
```

Out[27]:
```
list
```

Basic 'for' loop syntax (**mind the indentation, very important, usually 4 whitespaces or one \t**)

In [ ]:
```
for step in list_of_steps:
    #do whatever
```

The `range()` function is very handy in combination with for-loops. Let's create a loop that iterates through the numbers 0-20

In [29]:
```
for i in range(21):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

You can also iterate through an existing list, e.g. our list `concatenated_list` from above. Let's also define a simple counter, to which we add `1` in every round, thereby keeping track of the number of repititions in our loop:

In [30]:
```
counter = 0
for word in concatenated_list:
    counter = counter + 1
    print('Word %i: %s'%(counter,word))
```

```
Word 1: I
Word 2: FREAKING
Word 3: LOVE
Word 4: PYTHON
Word 5: BECAUSE
Word 6: IT
Word 7: IS
Word 8: SO
Word 9: MUCH
Word 10: FUN
```

Sometimes you want to iterate through a list and save some variable every round. An easy way to do this is to append a variable to a list in every iteration. For this we first define an empty list at the beginning of the loop and then append new values to it every iteration.

Say we want to iterate through the numbers 0-20 and record the sum of all previous numbers in every round. That would look as follows:

In [31]:
```
sum_list = [0]
for i in range(21):
    new_sum = i + sum_list[-1]
    sum_list.append(new_sum)
print(sum_list)
```

```
[0, 0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171,
190, 210]
```

Task: Create a loop that prints an increasing number of words of a sentence. You can use the sentence:

```
If I can do this I am mastering the basic syntax of python.
```

Tip: In order to iterate through the words of a string you first have to turn the string into a list. For this you can use the `split()` function from earlier in the tutorial, and split the string at each whitespace `' '` character.

The output should look like this:

In [114]:
```

```

```
['If']
['If', 'I']
['If', 'I', 'can']
['If', 'I', 'can', 'do']
['If', 'I', 'can', 'do', 'this']
['If', 'I', 'can', 'do', 'this', 'I']
['If', 'I', 'can', 'do', 'this', 'I', 'am']
['If', 'I', 'can', 'do', 'this', 'I', 'am', 'mastering']
['If', 'I', 'can', 'do', 'this', 'I', 'am', 'mastering', 'the']
['If', 'I', 'can', 'do', 'this', 'I', 'am', 'mastering', 'the', 'basic']
['If', 'I', 'can', 'do', 'this', 'I', 'am', 'mastering', 'the', 'basic', 'syntax
']
['If', 'I', 'can', 'do', 'this', 'I', 'am', 'mastering', 'the', 'basic', 'syntax
', 'of']
['If', 'I', 'can', 'do', 'this', 'I', 'am', 'mastering', 'the', 'basic', 'syntax
', 'of', 'python.']
```

Let's move on to tutorial 2 (https://github.com/tobiashofmann88/python_for_biologists/blob/master/tutorials/tutorial_2.ipynb)