M4453: Introduction to Molecular Systems Biotechnology

Learning Python Supporting Material - I

02-06.10.2015

1 Datatypes

Python has several native datatypes, the most important that we encountered are:

- Booleans: they have the values True or False and are the result of logic operations
- Numbers: can be integers (23) or floats (23.7)
- Strings: sequences of Unicode characters like 'Hello World'
- Lists: sequences of values (booleans, numbers, strings...)
- Tuples: immutable sequences of values
- Dictionaries: collection of key-value pairs

Important: if you divide two integers you will get an integer! Try in the Python console:

```
>>> 5/3
1
>>> 5.0/3
1.6666666666666667
```

Try also the other operators we used:

```
>>> 5 >= 3
True
>>> 5 == 3
False
>>> 8%2
0
>>> 8%3
2
>>> 8%4
0
>>> 2**3
8
>>> 5//3
1
>>> 6//3
2
```

Remember: comparison operators are particularly important! Is equal (==), is different (!=), is higher (>), is lower (<), is higher or equal (>=), is lower or equal (<=). The logical operators `and` and `or` are used to combine conditions:

```
>>> 4 > 2 and 3 <6
True
>>> 4 > 2 and 3 < 2
```

```
False
>>> 4 > 2 or 3 < 2
True
```

Let's now work on a dictionary containing the ages of some persons:

```
>>> ages = {'Mary':53, 'July':19, 'Mark':9, 'Bob':22, 'Jill':36, 'Patty':68, 'Peter
    ':25, 'Max':76, 'Fabrice':12, 'Marianne':15}
>>> ages['Patty']
68
```

2 Control Structures

We started with Python doing plain operations (function calling, variable assignment...) that are executed line by line, in a consequent order, by the interpreter. Can we have more control on the **flow** of the program we write? The answer is luckily yes: there are special "commands", called control structures, that allow you to make your code "dynamic".

2.1 if/else condition

The if/else structure is used when you want to test a particular True/False condition. A boolean can be the result of some **logical** operation. Execute now file `ages.py` or copy its content in the shell:

```
'''@author: antonella'''
if ages['Patty'] < 13:
    print 'Patty is a child'
elif ages['Patty'] < 18:
    print 'Patty is a teenager'
else:
    print 'Patty is an adult'
```

```
>>> execfile('ages.py')
Patty is an adult
```

The else clause is not mandatory: can you tell what happens in the example above removing the `else` condition?

2.2 for loops

You will often need to repeat some operations on, e.g., all the elements of a list. For loops are easily defined through an **index** that goes over a set of values. We will now use the `keys` method of a dictionary to obtain a list of the names of our dictionary:

```
>>> ages.keys()
['Fabrice', 'Jill', 'Peter', 'Marianne', 'Bob', 'Max', 'July', 'Patty', 'Mary', '
    Mark']
```

and will now put our previous if conditions in a loop over the content of the dictionary (file `loopages.py`):

```
'''@author: antonella'''
for k in ages.keys():
    if ages[k] < 13:
        print k,'is a child'
    elif ages[k] < 18:
        print k,'is a teenager'
    else:
        print k,'is an adult'
```

```
>>> execfile('loopages.py')
Fabrice is a child
Jill is an adult
Peter is an adult
Marianne is a teenager
Bob is an adult
Max is an adult
July is an adult
Patty is an adult
Mary is an adult
Mark is a child
```

2.3 while loops

A while loop defines a set of instructions that are repeated until a certain condition is met. We can e.g. replicate the previous for loop as (file `loopages2.py`):

```
'''@author: antonella'''
akeys = ages.keys()
i = 0
while i < len(akeys):
    k = akeys[i]
    if ages[k] < 13:
        print k,'is a child'
    elif ages[k] < 18:
        print k,'is a teenager'
    else:
        print k,'is an adult'
    i += 1
```

```
>>> execfile('loopages2.py')
Fabrice is a child
Jill is an adult
Peter is an adult
Marianne is a teenager
Bob is an adult
Max is an adult
July is an adult
Patty is an adult
Mary is an adult
Mark is a child
```

This is, of course, a very bad example! Why use more lines of code when you could simply define a for loop? There might be cases, anyway, when you will have to do something similar, so keep it in mind, and remember that at the end it's always your choice and your style, and as long as it works, it's ok!

Here we show another while loop example, a bit more meaningful (file `sumages.py`):

```
'''@author: antonella'''
akeys = ages.keys()
i = 0
asum = 0
while asum < 100:
    asum += ages[akeys[i]]
    i += 1
    if i >= len(akeys):
        print 'index out of scope!'
```

```
        break
print asum
```

```
>>> execfile('sumages.py')
110
```

Do you understand what is happening here? Do you remember what `break` is for?
Finally, remember that a loop like

```
while(TRUE){print("Hello world!")}
```

will go on forever! This happens if e.g. you forget to increment the `i` counter in the `loopages2.py` while loop! Be careful, anyway you can always stop any process in the shell by pressing CTRL+C.

3 Functions

Functions are a very convenient way to reuse code. Let's say you have to compute the average value for numbers stored in a list. You can either write code lines for every set of data like in `fun1.py`:

```
'''@author: antonella'''
nlist = ages.values()
avg = 0.0
for n in nlist:
    avg += n
avg = avg/len(nlist)
print avg
```

```
>>> execfile('fun1.py')
33.5
```

Or you can define a function with variable arguments and call it several times like in `fun2.py` with the different lists as arguments:

```
'''@author: antonella'''

def getAvg(nlist):
    '''function to return the average of a list of numbers'''
    avg = 0.0
    for n in nlist:
        avg += n
    avg = avg/len(nlist)
    return avg

teens = []
adults = []
kids = []
allps = ages.values()

for a in allps:
    if a < 13:
        kids.append(a)
    elif a <18:
        teens.append(a)
    else:
        adults.append(a)

print 'The average age is ',getAvg(allps)
```

```
print 'The average teenagers age is ',getAvg(teens)
print 'The average adults age is ',getAvg(adults)
print 'The average kids age is ',getAvg(kids)
```

```
>>> execfile('fun2.py')
The average age is  33.5
The average teenagers age is  15.0
The average adults age is  42.7142857143
The average kids age is  10.5
```

4 Exercises

- In Section 2.1 we showed an if/else examples. Rewrite the conditions to start checking if the person is an adult. You should get the same result in the end!
- Write a code that prints out only the numbers between 1 and 20 that are multiple of 3. Tip: the modulo (integer remainder) operator is %.
- The Fizz Buzz test: Write a program that prints the numbers from 1 to 100. But for multiples of three print 'Fizz' instead of the number and for the multiples of five print 'Buzz'. For numbers which are multiples of both three and five print 'FizzBuzz'.