Introduction to python for scientific computing

- Lecture 7

No lecture on Wednesday, May 4th

- Postpone to Friday, 13th
- Same place, same time

Start by doing today's quiz (Review Day 3)

Review: In what ways does the type of an object matter?

In [10]:
```
row = 'sofa|2000|buy|Uppsala'
fields = row.split('|')
price = int(fields[1])
if price == 2000:
    print('The price is a number!')
if price == '2000':
    print('The price is a string!')
```

```
The price is a number!
```

In [11]:
```
print(sorted([ 2000,   30,   100 ]))
print(sorted(['2000', '30', '100']))
# Hint: is `'30' > '2000'`?
```

```
[30, 100, 2000]
['100', '2000', '30']
```

In what ways does the type of an object matter?

- Each type store a specific type of information
    - `int` for integers,
    - `float` for floating point values (decimals),
    - `str` for strings,
    - `list` for lists,
    - `dict` for dictionaries.
- Each type supports different operations, functions and methods.

```
In [12]: 30 > 2000

Out[12]: False

In [13]: '30' > '2000'

Out[13]: True

In [14]: 30 > int('2000')

Out[14]: False
```

```
In [15]: 'ACTG'.lower()

Out[15]: 'actg'

In [17]: int("3")

Out[17]: 3
```

```
In [18]: float('2000')
Out[18]: 2000.0

In [19]: float('0.9')
Out[19]: 0.9

In [20]: float('1e9')
Out[20]: 1000000000.0

In [21]: float('1e-2')
Out[21]: 0.01

In [ ]: int('2000')

In [ ]: int('1.5')

In [ ]: int('1e9')
In [22]: bool(1)

Out[22]: True

In [23]: bool(0)

Out[23]: False

In [ ]: bool('1')

In [24]: bool('0')

Out[24]: True

In [25]: bool('')

Out[25]: False

In [26]: bool({})

Out[26]: False
```

In [27]:
```python
values = [1, 0, '', '0', '1', [], [0]]
for x in values:
    if x:
        print(repr(x), 'is true!')
    else:
        print(repr(x), 'is false!')
```

```
1 is true!
0 is false!
'' is false!
'0' is true!
'1' is true!
[] is false!
[0] is true!
```

```
In [28]: list("hello")

Out[28]: ['h', 'e', 'l', 'l', 'o']

In [29]: str(['h', 'e', 'l', 'l', 'o'])

Out[29]: "['h', 'e', 'l', 'l', 'o']"

In [31]: '_'.join(['h', 'e', 'l', 'l', 'o'])

Out[31]: 'h_e_l_l_o'
```

Container types, when should you use which?

- **lists**: when order is important
- **dictionaries**: to keep track of the relation between keys and values
- **sets**: to check for membership. No order, no duplicates.

```
In [32]: genre_list = ["comedy", "drama", "drama", "sci-fi"]
         genre_list
```

```
Out[32]: ['comedy', 'drama', 'drama', 'sci-fi']
```

```
In [33]: genres = set(genre_list)
         genres
```

```
Out[33]: {'comedy', 'drama', 'sci-fi'}
```

```
In [34]: 'drama' in genres
```

```
Out[34]: True
```

```
In [ ]: genre_counts = {"comedy": 1, "drama": 2, "sci-fi": 1}
        genre_counts
```

```
In [ ]: movie = {"rating": 10.0, "title": "Toy Story"}
        movie
```

What is a function?

- A named piece of code that performs a specific task
- A relation (mapping) between inputs (arguments) and output (return value)

```python
def hello_function(number):
    # print the user input
    print(number)
    number += 2
    return 2
```

TODAY

- More on functions: keyword arguments, return statement...
- Reusing code:
    - comments and documentation
    - importing modules: using libraries
- Pandas - explore your data!

More on functions: scope - global variables and local function variables

```
In [1]: movies = ['Toy story', 'Home alone']

        def some_thriller_movies():
            return ['Fargo', 'The Usual Suspects']

        movies = some_thriller_movies()
        print(movies)
```

```
['Fargo', 'The Usual Suspects']
```

```
In [40]: movies = ['Toy story', 'Home alone']

         def add_a_movie(list_of_movies):
             list_of_movies.append("Thor")
             return list_of_movies

         add_a_movie(movies)
         print(movies)
```

```
['Toy story', 'Home alone', 'Thor']
```

Takeaway message: be careful with your variable names!

Also, global variables are usually not a good idea

More on functions

A function that counts the number of occurences of `'C'` in the argument string.

In [41]:
```
def cytosine_count(nucleotides):
    count = 0
    for x in nucleotides:
        if x == 'c' or x == 'C':
            count += 1
    return count

count1 = cytosine_count('CATATTAC')
count2 = cytosine_count('tagtag')
print(count1, count2)
```

```
2 0
```

```
In [43]: total_count = cytosine_count('catattac') + cytosine_count('tactactac')
         print(total_count)
```

```
5
```

```
In [44]: def print_cytosine_count(nucleotides):
             count = 0
             for x in nucleotides:
                 if x == 'c' or x == 'C':
                     count += 1
             print(count)

         print_cytosine_count('CATATTAC')
         print_cytosine_count('tagtag')
```

```
2
0
```

```
In [45]: print_cytosine_count('catattac') + print_cytosine_count('tactactac')
```

```
2
3

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [45], in <cell line: 1>()
----> 1 print_cytosine_count('catattac') + print_cytosine_count('tactactac')

TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

In [76]:
```
l = [1, 2, 3]
l.append(4)
print(l)
```
```
[1, 2, 3, 4]
```

Keyword arguments

- A way to give a name explicitly to a function for clarity

```
In [55]: sorted('file', reverse=True)
```

```
Out[55]: ['l', 'i', 'f', 'e']
```

```
In [61]: def split(sep, maxsplit, ...)

         attribute = 'gene_id "unknown gene"'
         attribute.split(sep=' ', maxsplit=1)
```

```
Out[61]: ['gene_id', '"unknown gene"']
```

```
In [60]: # print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
         print('x=', end=' ')
         print('1', end="_")
```

```
x= 1_
```

Keyword arguments

- Order of keyword arguments do not matter

```
open(file, mode='r', encoding=None) # some arguments omitted
```

- These mean the same:

```
open('files/recipes.txt', 'w', encoding='utf-8')

open('files/recipes.txt', mode='w', encoding='utf-8')

open('files/recipes.txt', encoding='utf-8', mode='w')
```

Defining functions taking keyword arguments

- Just define them as usual:

```
In [64]: def format_sentence(subject, value, end):
             return 'The ' + subject + ' is ' + value + end

         print(format_sentence('lecture', 'ongoing', '.'))

         print(format_sentence('lecture', 'ongoing', end='!'))

         print(format_sentence(subject='lecture', end='...', value='ongoing'))
```

```
The lecture is ongoing.
The lecture is ongoing!
The lecture is ongoing...
```

```
In [65]: print(format_sentence(subject='lecture', 'ongoing', '.'))
```

```
  Input In [65]
    print(format_sentence(subject='lecture', 'ongoing', '.'))
                                                          ^
SyntaxError: positional argument follows keyword argument
```

- Positional arguments comes first, keyword arguments after!

Defining functions with default arguments

In [68]:
```python
def format_sentence(subject, value, end='.'):
    return 'The ' + subject + ' is ' + value + end

print(format_sentence('lecture', 'ongoing'))

#print(format_sentence('lecture', value='ongoing', end='...'))
```

```
The lecture is ongoing.
```

Defining functions with optional arguments

- Convention: use the object `None`

In [69]:
```python
def format_sentence(subject, value, end='.', second_value=None):
    if second_value is None:
        return 'The ' + subject + ' is ' + value + end
    else:
        return 'The ' + subject + ' is ' + value + ' and ' + second_value + end

print(format_sentence('lecture', 'ongoing'))

print(format_sentence('lecture', 'ongoing',
                      second_value='self-referential', end='!'))
```

```
The lecture is ongoing.
The lecture is ongoing and self-referential!
```

Small detour: Python's value for missing values: `None`

- Default value for optional arguments
- Implicit return value of functions without a `return`

```
In [70]: bool(None)
```

```
Out[70]: False
```

```
In [71]: None == False, None == 0
```

```
Out[71]: (False, False)
```

```
In [74]: def print_something(input1=None):
             if input1 is None:
                 pass
             else:
                 print(input1)

         print_something("a")
```

```
a
```

```
In [72]: if None:
             print('None is true')
         else:
             print('None is not true')
```

```
None is not true
```

In [75]:

```python
values = [None, 1, 0, '', '0', '1', [], [0]]
for x in values:
    if x is None:
        print(repr(x), 'is None')
    if not x:
        print(repr(x), 'is false')
    if x:
        print(repr(x), 'is true')
```

```
None is None
None is false
1 is true
0 is false
'' is false
'0' is true
'1' is true
[] is false
[0] is true
```

Exercise

Create a movie picker function. The function will pick the first movie that fits the user's requirements and print its title. The user can choose to pick a movie based on one or more of the four requirements `year` , `genre` , `minimal rating` or `maximal rating` .

```
>>> pick_movie(genre="Drama")

The Paths of Glory

>>> pick_movie(year=2001)

Donnie Darko

>>> pick_movie(rating_min=8)

Paths of Glory

>>> pick_movie(year=2009, genre="Mystery")

The Secret in Their Eyes
```