Python Workshop. Day 1

CompBioLab

- Python functions: Build-in and custom functions

1. Overview of Built-in functions
2. Filter, filterfalse and list comprehensions
3. Nice functions from libraries
4. How to make a custom function
5. Python Tutor
6. Example of debugging with jupyter-notebook

-Numpy

1. Why we use Numpy?
2. NumPy.array()
3. Data selection
4. Basic Numpy operations

Python functions: Built-in and custom functions

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
- Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.
- Calling a function: Returns = Function(Arguments)

A **method** is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:

1. A method is implicitly passed the object on which it was called.
2. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).

- Calling a method: *Object.Method(Arguments)* (the methods modifies the objects, do not returns a new one)

Python functions: Built-in

The Python interpreter has a number of functions and types built into it that are always available.

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

https://docs.python.org/3/library/functions.html

Python functions: Built-in

Interesting “Build-in” functions

- *print()* (In python 3.x, print is a function and needs a paranthesis to work)
- *format()* (Replaces the old formatting modulo %. https://pyformat.info/ (nice explanation of how format language works)). Format is also an string Method
- *len()*. Return the length (the number of items) of an object.
- *open()*. Open *file* and return a corresponding file object.

Open modes:

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open a disk file for updating (reading and writing)
`'U'`	universal newlines mode (deprecated)

More open options and functions in libraries (Day 2).

Python functions: Built-in

Interesting “Build-in” functions

- *zip()* Returns an iterator of tuples, where the *i*-th tuple contains the *i*-th element from each of the argument sequences or iterables.

  Eg:
  ```
  >>> x = [1, 2, 3]
  >>> y = [4, 5, 6]
  >>> zipped = zip(x, y)
  >>> list(zipped)
  [(1, 4), (2, 5), (3, 6)]
  ```

- *enumerate()* Return an enumerate object. *iterable* must be a sequence, an iterator, or some other object which supports iteration

  Eg:
  ```
  >>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
  >>> list(enumerate(seasons))
  [(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
  >>> list(enumerate(seasons, start=1))
  [(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
  ```
  a list created with enumerate has an order, but dictionaries not!

Python functions: Built-in

Interesting “Built-in” functions

- *map()* Return an iterator that applies *function* to every item of *iterable*, yielding the results.

  Eg:

```
>>> items = [ 2, 4, 6, 8, 10]
>>> half = []
>>> for x in items:
        half.append(x / 2)
>>> half
[1, 2, 3, 4, 5]
```

```
>>> items = [2, 4, 6, 8, 10]
>>> map(lambda x: x/2, items)
[1, 2, 3, 4, 5]
```

```
>>> items = [2, 4, ‘Lasagna’, 8, 10]
>>> map(lambda x: x/2, items)
CombinationError, unsupported dish,
‘Lasagna’ is not divisible
```

- *filter()* Construct an iterator from those elements of *iterable* for which *function* returns *True*.

  Eg:

```
>>> items = [1, 2, 3, 4, 5, 6]
>>> even_numbers = []
>>> for num in items:
        if num % 2 == 0:
            even_numbers.append(num)
>>> even_numbers
[2, 4, 6]
```

```
>>> items = [1, 2, 3, 4, 5, 6]
>>> even_numbers = filter(lambda x: x % 2 == 0, items)
>>> even_numbers
[2, 4, 6]
```

https://docs.python.org/3/library/itertools.html#itertools.filterfalse

Python functions: list comprehensions

```
>>> items = [1, 2, 3, 4, 5, 6]
>>> even_numbers = []
>>> for num in items:
        if num % 2 == 0:
            even_numbers.append(num)
>>> even_numbers
[2, 4, 6]
```

```
>>> items = [1, 2, 3, 4, 5, 6]
>>> even_numbers = filter(lambda x: x % 2 == 0, items)
>>> even_numbers
[2, 4, 6]
```


List Comprehesions

```
>>> items = [1, 2, 3, 4, 5, 6]
>>> even_numbers = [i for i in items if i % 2 ==0]
>>> even_numbers
[2, 4, 6]
```

```
numbers = [1, 2, 3, 4, 5]

doubled_odds = []
for n in numbers:
    if n % 2 == 1:
        doubled_odds.append(n * 2)


~
```

Notebook 1

http://treyhunner.com/2015/12/python-list-comprehensions-now-in-color/

Python functions: from libraries

-math
- math.fabs() #Absolut value
- math.exp() #Exponential
- math.log() #Logarithmic
- math.e() # e number
- math.pi() # pi number

-os
- os.system() #Execute shell commands
- os.getcwd() #Get the current working directory
- os.listdir() #Get all the files in the directory
- os.mkdir() # Shell mkdir

-sys: sys.argv[] #List of command line arguments passed to a Python script. argv[0] is the script name.

"I know, this is not a function :p"

-glob #glob.glob() #Return a possibly-empty list of path names that match pathname. Can be used re symbols.

ex: `glob.glob('../../Tools/*/*.gif')`

-itertools
- itertools.combinations() #Return subsequences of elements from the input iterable
- itertools.imap() #Like map() but with multiple iterables

-time #provides various time-related functions

-datetime #manipulation of dates and times

More: Shutil, random, re, cython and theano

Python functions: How to make a custom function

Output = Custom_func(arguments)

Ex:

```
def word_information(word):
    vocals = len([x for x in word if x in ['a','e','i','o','u']])
    consonants = len(word) - vocals
    print('The word has {} vocals and {} consonants'.format(vocals,consonants))

word_information('CompBioLab')
```

```
The word has 4 vocals and 6 consonants
```

Ex2:

```
def n_primes(limit):
    noprimes = [j for i in range(2, limit) for j in range(i*2, limit, i)]
    primes = [x for x in range(2, limit) if x not in noprimes]
    return primes
```

```
print(n_primes(1000))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 2
41, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 38
3, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523,
541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 67
7, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839,
853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
```

Python functions: How to make a custom function

Output = Custom_func(arguments)

Arguments:

- Can be any type of variable.
- The name of this arguments are only valid inside the function, and any change to this do not affect the original variable.
- If the variable is defined with a default value (‘argument = 2’), this will be an optional argument, and if this is not passed, the default value will be the defined in the function (see example in the Notebook 1)

- If you are having problems, visit http://pythontutor.com/visualize.html#mode=edit

- Use the “%%timeit” (jupyter-notebook function) to chose wich solution is faster!!

NumPy

- **Why NumPy is necessary?**
  - Python lists are only 1D (vectors)
  - NumPy arrays are multi-dimensional (matrices)
  - Python alone have no idea how to do calculations between lists

- **Why NumPy is better for computation?**
  - It is convenient for doing operation over arrays
  - It is faster
  - It requires less memory

Python

x

$$\begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

NumPy

M

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & \dots \end{pmatrix}$$

NumPy reference documentation: https://docs.scipy.org/doc/

NumPy

- **1D NumPy array:**

```
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
>>> a.ndim
1
>>> a.shape
(4,)
>>> len(a)
4
```

- **2D, 3D NumPy arrays:**

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]])    # 2 x 3 array
>>> b
array([[0, 1, 2],
       [3, 4, 5]])
>>> b.ndim
2
>>> b.shape
(2, 3)
>>> len(b)     # returns the size of the first dimension
2

>>> c = np.array([[[1], [2]], [[3], [4]]])
>>> c
array([[[1],
        [2]],

       [[3],
        [4]]])
>>> c.shape
(2, 2, 1)
```

Numpy

- **NumPy structured arrays:**

```
name = ['Alice', 'Bob', 'Cathy', 'Doug']
age = [25, 45, 37, 19]
weight = [55.0, 85.5, 68.0, 61.5]
```

```
data = np.zeros(4, dtype={'names':('name', 'age', 'weight'),
                          'formats':('U10', 'i4', 'f8')})
print(data)
```

```
[('', 0,  0.) ('', 0,  0.) ('', 0,  0.) ('', 0,  0.)]
```

```
data['name'] = name
data['age'] = age
data['weight'] = weight
print(data)
```

```
[('Alice', 25,  55. ) ('Bob', 45,  85.5) ('Cathy', 37,  68. )
 ('Doug', 19,  61.5)]
```

```
# Get names where age is under 30
data[data['age'] < 30]['name']
```

```
array(['Alice', 'Doug'],
      dtype='<U10')
```

Numpy

- **NumPy data selection: indexing**

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[0], a[2], a[-1]
(0, 2, 9)
```

```
>>> a = np.diag(np.arange(3))
>>> a
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
>>> a[1, 1]
1
>>> a[2, 1] = 10 # third line, second column
>>> a
array([[ 0,  0,  0],
       [ 0,  1,  0],
       [ 0, 10,  2]])
>>> a[1]
array([0, 1, 0])
```

Numpy

- **NumPy data selection: slicing**

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2:9:3] # [start:end:step]
array([2, 5, 8])
```

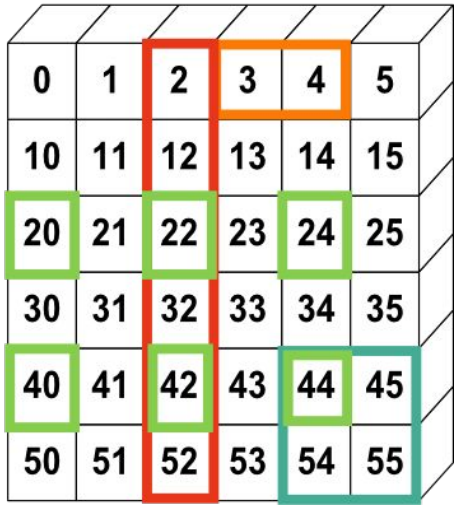
```
>>> a[1:3]
array([1, 2])
>>> a[::2]
array([0, 2, 4, 6, 8])
>>> a[3:]
array([3, 4, 5, 6, 7, 8, 9])
```

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



Numpy

- **Fancy indexing and masks**

```
>>> np.random.seed(3)
>>> a = np.random.randint(0, 21, 15)
>>> a
array([10,  3,  8,  0, 19, 10, 11,  9, 10,  6,  0, 20, 12,  7, 14])
>>> (a % 3 == 0)
array([False,  True, False,  True, False, False, False,  True, False,
        True,  True, False,  True, False, False], dtype=bool)
>>> mask = (a % 3 == 0)
>>> extract_from_a = a[mask] # or,  a[a%3==0]
>>> extract_from_a           # extract a sub-array with the mask
array([ 3,  0,  9,  6,  0, 12])
```

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```