Introduction to Python

Session 05

List comprehensions

Iterables

An object that can be iterated upon returning its members one at a time.

- Lists
- Tuples
- Dictionaries
- Sets
- Strings

In python everything is an object

are containers

```
for element in iterable:
    print(element)
```

TypeError: 'float' object is not iterable

upf.

Built-in methods applicable to iterators

- **sum**
- **max**
- **min**

```
>>> max([1,39,10,0,-3])
39
>>> max(set([1,39,10,0,-3]))
39
>>> max("adfadijaskf")
's'
```

if it's a dictionary you get the max value of the key

gives you the highest letter with ASCI code

Built-in methods applicable to iterators

- Sequence functions
  - **map**
  - **filter**
  - **zip**

upf.

Built-in methods applicable to iterators

- **map:** Applies a **function** to every item in an iterable and returns an iterator with the results.

```
map(func, iterable)--> map object
```

Example: convert a list of integers to a list of strings:

```
>>> my_number_list = [1,2,3,4,5,6,7]
>>> map(str, my_number_list)
<map object at 0x2b6a72a48790>
>>> for element in map(str, my_number_list):
...     print(element)
...
```

in this case the funciton is str (converts somehting to string)

as it is an iterator we can loop over it

upf.

Built-in methods applicable to iterators

- **map:** Applies a **function** to every item in an iterable and returns an iterator with the results.

```
map(func, iterable)--> map object
```

Example: Apply a mathematical function to each element of a list

```
>>> my_list = [1,2,3,4,5,6,7,8,9,10]
>>> import math
>>> for element in map(math.sin, my_list):
...     print(element)

0.8414709848078965
0.9092974268256817
0.1411200080598672
…
```

Built-in methods applicable to iterators

- **map:** Applies a **function** to every item in an iterable and returns an iterator with the results.

```
map(func, iterable)--> map object
```

Example: look for the maximum value of several lists

```
>>> my_lists = [ [1,2,3], [4,5,6], [1,0,10] ]
>>> list(map(max, my_lists))
[3, 6, 10]
```

upf.

Built-in methods applicable to iterators

- **map:** Applies a **function** to every item in an iterable and returns an iterator with the results.

```
map(func, iterable)--> map object
```

What if the function I need to apply does not exist?

1) Implement it as a new function.

a funciton that is created and used, it is not stored

2) Create a **lambda function**: **Functions syntactically restricted to a single expression**.

```
lambda x: x+1
```

Lambda functions

- Function restricted syntactically to a single expression.
- **Anonymous** functions (not linked to a name)
- Commonly used combined with other functions, such as map.

```
lambda x:  x+1

lambda x,y:  x+y/2.0

lambda l: l[1]
```

upf.

Built-in methods applicable to iterators

- **map:** Applies a **function** to every item in an iterable and returns an iterator with the results.

`map(func, iterable)--> map objectt`

Example: Apply a lambda function

```
>>> my_list = [1,2,3,4,5]
>>> map(lambda x: x-1, my_list)
[0, 1, 2, 3, 4]
```

substract one from each of the elements

Built-in methods applicable to iterators

- Sequence functions
  - **map**
  - **filter**
  - **zip**

upf.

Built-in methods applicable to iterators

- **filter:** Return those items of sequence for which function(item) is true
  - If function is None, return the items that are true.
  - If sequence is a tuple or string, return the same type, else return a list.

```
filter(function or None, sequence)
```

Built-in methods applicable to iterators

- **filter:** Return those items of sequence for which function(item) is true

`filter(function or None, sequence)`

Example: Filter a list of integers and get only the even numbers.

```
>>> my_list = [1,10,5,3,1,4,8,40,11,15]

>>> list( filter(lambda x: x%2==0, my_list) )
[10, 4, 8, 40]
```

Built-in methods applicable to iterators

- **zip:** Return an iterator of tuples, where each tuple contains the i-th element from each of the argument sequences.

  The returned iterator ends to the length to the length of the shortest argument sequence.

zip(*iterables)

```
>>> list_a = [1,2,3,4,5]
>>> list_b = ["a","b","c","d"]
>>> list(zip(list_a, list_b))
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]
```

unifying the first element of the list with the first of the other list

upf.

List comprehensions

A more versatile and readable alternative

```
my_list =  [function(x) for x in iterable]
```

Example: create a new List with the logarithm of numbers 1 to 9:

```
>>> [ math.log(x) for x in range(1,10) ]
[0.0, 0.6931471805599453, 1.0986122886681098,
1.3862943611198906, 1.6094379124341003, 1.791759469228055,
1.9459101490553132, 2.0794415416798357, 2.1972245773362196]
```

the last number(10) is never included

upf.

List comprehensions

Another way of building a List object:

```
my_list =  [function(x) for x in iterable]
```

Example: create a new List with strings with 2 decimal positions of with the logarithm of numbers from 1 to 9:

put at the beggining the string formatting

```
>>> ["%.2f"%math.log(x) for x in range(1,10)]
['0.00', '0.69', '1.10', '1.39', '1.61', '1.79', '1.95',
'2.08', '2.20']
```

List comprehensions

Another way of building a List object:

```
my_list =  [function(x) for x in iterable if
                    condition]
```

Example: create a new List with strings with 2 decimal positions of with the logarithm of numbers from 1 to 50 that are divisible by 5

```
>>> ["%.2f"%math.log(x) for x in range(1,51) if x%5==0]
['1.61', '2.30', '2.71', '3.00', '3.22', '3.40', '3.56',
'3.69', '3.81', '3.91']
```

List comprehensions

Another way of building a List object:

```
my_list =  [function(x) for x in iterable if
                    condition]
```

Example: create a list with all identifiers from a FASTA file for proteins ending with some residues:

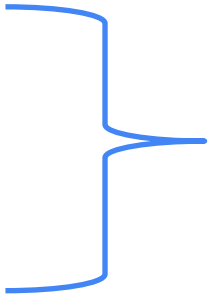
```
>>> my_list = [ seq_id for seq_id, sequence in
FASTA_iterator("my_fasta_file.fa") if sequence[-3:]=="GNN"]
```

I'm calling seq_id instead of x.
it returns a sequence that only if sequence starts at a position and is GNN.

upf.

Built-in summary

- Python basic objects:
  - int
  - float
  - boolean
  - None
  - String
  - List
  - Dictionary
  - Set



- Python built-in functions:
  - Convert between types (int(), float(), bool(), str())…
  - Functions applicable to iterables:
    - map, filter, zip, max, min, sum
  - Other functions:
    - input, type, range, sorted, …

upf.

Some built-in functions

- **input(message)**
  - Read a **string** from standard input (keyboard).

```
>>> number = input("Please insert a number: ")
Please insert a number: 4
>>> print(number)
4
>>> type(number)
<type 'str'>
```

you have a promopt in which you will type 4.

this is to test function

The return value is a **String**!

Some built-in functions

- Conversions: to integer and float

```
>>> int("10")
10
>>> float(10)
10.0
```

- To Boolean:

```
>>> bool(10)
True
>>> bool(-3.14)
True
>>> bool("False")
True
>>> bool(0)
False
```

Any int is True except for 0

Everything is converted to True except value 0

upf.

Some built-in functions

- Eval: Dynamically evaluate expressions

```
>>> eval("False")
False

>>> eval("10")
10

>>> type(eval("10"))
<type 'int'>

>>> s = "hello"
>>> eval("s")
'hello'
```

not recomended to use,

upf.

More on list creation

start and step is included, but not stop

- **range**

```
range([start,] stop[, step])
```

  - Returns a virtual sequence of numbers from start to stop by step.
  - start (!) defaults to 0.
  - When step is given, it specifies the increment (or decrement).
  - The endpoint is omitted.

upf.

More on list creation

- **range**

```
range([start,] stop[, step])
```

```
>>> range(10)
range(0, 10)

>>> list( range(10) )
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


>>> list( range(1,10,2) )
[1, 3, 5, 7, 9]
```

upf.

Built-in summary

- To see all available built-in functions:

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

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

upf.

Exercises. Session 5.

PRACTICE LIST COMPREHENSIONS
ONE SINGLE LINE OF CODE COULD BE DONE!

Create a python script called **`<uID>_S05.py`** with:

Note: Use the `FASTA_iterator` function created in last exercises.

1) Repeat the same exercises proposed in session 2 but using the FASTA_Iterator function created in session 4 to read the FASTA files.

2) A function that, given a multiline FASTA file, returns the length of the sequence with the maximum length

```
get_max_sequence_length_from_FASTA_file( fasta_filename )
```

3) A function that, given a multiline FASTA file, returns the length of the sequence with the minimum length

```
get_min_sequence_length_from_FASTA_file ( fasta_filename )
```

4) A function that, given a FASTA file, returns a list of tuples (identifier, sequence) corresponding to the sequence(s) with maximum length. The list must be sorted by the identifier (case insensitive sorted).

```
get_longest_sequences_from_FASTA_file( fasta_filename )
```

upf.

Exercises. Block 1. Part 5.

5) A function that, given a FASTA file, **returns a list of tuples** (identifier, sequence) corresponding to the sequence(s) with minimum length. The list must be sorted by the identifier (case insensitive sorted).

```
get_shortest_sequences_from_FASTA_file( fasta_filename )
```

6) A function that, given a protein FASTA file, returns a dictionary with the molecular weights of all the proteins in the file. The dictionary keys must be the protein identifiers and the associated values must be a float corresponding to the molecular weight.

```
get_molecular_weights( fasta_filename )
```

VALUES CORRESPONDING TO MOLEUCLAR WEITHG

7) A function that, given a protein FASTA file, returns a tuple with (identifier, sequence) of the protein with the lowest molecular weight. If there are two or more proteins having the minimum molecular weight, just return the first one.

```
get_sequence_with_min_molecular_weight( fasta_filename )
```

8) A function that, given a protein FASTA file, returns the mean of the molecular weights of all the proteins

```
get_mean_molecular_weight( fasta_filename )
```

upf.