Python intro - Lab 1

Objectives:

- How to run python?
- Basic data types

1 Introduction

Python supports multiple programming paradigms, including object-oriented, procedural and functional styles.

- **Proper indentation is a must**. Blocks are represented with indentation, there is no need to use curly braces.
- You don't need to define the type of variables.
- You don't need to add semicolon at the end of the statements.
- Python is case sensitive
- A minimum set of good practices:
  - Variable, functions, methods, packages and modules: *lower_case_with_underscores*
  - Classes and exceptions: *CapWords*
  - Protected methods and internal functions: *_single_leading_underscore(self, ...)*
  - Constants: *ALL_CAPS_WITH_UNDERSCORE*

There are several ways to run a python code:

- Interactively via an interpreter: $ python
- Execute a script call from the command line: $ python your_file.py
- Use an IDE (Integrated development environment), like PyCharm, VisualStudio
- Use Jupyter notebooks or Jupyter lab

Observation: There are differences between python2 and python 3. Check your version.

2 Basics

Try different **arithmetic operators** and check the type of the variables: (use q to exist help)

```
$ python
Python 3.8.8 (default, Apr 13 2021, 19:58:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type ”help”, ”copyright”, ”credits” or ”license” for more information.
>>> a=2
>>> 2+3
5
>>> 2+”aaa”
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> type(a)
<class 'int'>
>>> type("aaa")
<class 'str'>
>>> help(a)

>>> 10/2
5.0
>>> 10%2
0
>>> 10%3
1
>>> 10/3
3.3333333333333335
>>> int(10/3)
3
>>> 2**3
8
>>> a=10
>>> b=12.2
>>> type(b)
<class 'float'>
```

Try different **comparison operators** (pay attention to True and False)

```
>>> a=10
>>> b=12.2
>>> type(b)
<class 'float'>
>>> a<b
True
>>> a==b
False
>>> a!=b
True
>>> a<=2
False
```

Try **logical operators**:

```
>>> a,b
(10, 12.2)
>>> not (a<2)
True
>>> a<2 and b>10
False
>>> a<2 or b>10
True
```

Printing: one variable, several variable, with/without output formatting

```
>>> print(a)
10
>>> print('a is ', a)
a is  10
>>> print(a, b, 12)
10 12.2 12
>>> print("a={}, b={}".format(a,b))
```

```
a=10, b=12.2
>>>print(f'a={a}, b={b}")
  File "<stdin>", line 1
    print(f'a={a}, b={b}")
                         ^

SyntaxError: EOL while scanning string literal
>>>print(f'a={a}, b={b}')
a=10, b=12.2
>>>print("a= %i, b=%.2f" % (a,b))
a= 10, b=12.20
```

Working with strings:

```
>>> a="hello world"
>>> b="in year 2022"
>>> a+b
'hello worldin year 2022'
>>> "world" in a
True
>>> "world" in b
False
>>> a.replace("hello", "see you")
'see you world'
>>> a
'hello world'
>>> len(a)
11
>>> a.split(" ")
['hello', 'world']
>>> "_".join(["I", "am", "a", "student"])
'I_am_a_student'
>>> a.upper()
'HELLO WORLD'
>>> a
'hello world'
```

2.1 Collections of data

Built-in data types to store collections of data: List, Tuple, Set, Dictionary.

Lists:

- a list can contain different data types
- lists allow duplicates
- the elements of a list can be changed

Try some examples with lists:

```
>>> a=[1,22,3,0.1]
>>> a
[1, 22, 3, 0.1]
>>> type(a)
<class 'list'>
>>> a=[1,22,3,"0.1"]
>>> a
[1, 22, 3, '0.1']
>>> len(a)
4
```

```
>>> a+[1,2]    #concatenate two lists
[1, 22, 3, ’0.1’, 1, 2]
>>> 1 in a  #test membership
True
```

Access the elements of a list: index, negative indexing, range of indexing (slicing)

```
>>> a=[”python”, ”java”, ”C”, ”html”]
>>> a[0]
’python’
>>> a[−1]
’html’
>>> a[−2]
’C’
>>> a[0:2]
[’python’, ’java’]
>>> a[:2]
[’python’, ’java’]
>>> a[2:−1]
[’C’]
>>> a[2:]
[’C’, ’html’]
```

Change the elements of a list:

```
>>> a[1]=”scala”
>>> a
[’python’, ’scala’, ’C’, ’html’]
```

List methods:

append() Adds an element at the end of the list

clear() Removes all the elements from the list

copy() Returns a copy of the list

count() Returns the number of elements with the specified value

extend() Add the elements of a list (or any iterable), to the end of the current list

index() Returns the index of the first element with the specified value

insert() Adds an element at the specified position

pop() Removes the element at the specified position

remove() Removes the item with the specified value

reverse() Reverses the order of the list

sort() Sorts the list

```
>>> a
[’python’, ’scala’, ’C’, ’html’]
>>> a.append(”sql”)
>>> a
[’python’, ’scala’, ’C’, ’html’, ’sql’]
>>> a.append(”sql”)
>>> a
[’python’, ’scala’, ’C’, ’html’, ’sql’, ’sql’]
>>> a.count(”sql”)
2
```

```
>>> a.index(”sql”)
4
>>> a.pop(−1)
'sql'
>>> a
['python', 'scala', 'C', 'html', 'sql']
>>> a.remove(”scala”)
>>> a
['python', 'C', 'html', 'sql']
>>> a.sort()
>>> a
['C', 'html', 'python', 'sql']
```

Min, max sum on lists:

```
>>> a=[1,3,2,4]
>>> max(a)
4
>>> min(a)
1
>>> sum(a)
10
>>> a=[[1,2], [0,10], [9, 1]]
>>> len(a)
3
>>> min(a)
[0, 10]
>>> min(a, key = lambda x: x[0])
[0, 10]
>>> min(a, key = lambda x: x[1])
[9, 1]
```

What is lambda expression? A function without a name

```
>>> f = lambda x : x[0]
>>> f
<function <lambda> at 0x7fc3125ae310>
>>> f([10,20])
10
```

List comprehension: shorter syntax when you want to create a new list based on the values of an existing list.

```
>>> a=[1,10,2,3]
>>> [e+1 for e in a]
[2, 11, 3, 4]
>>> [0 for e in a]
[0, 0, 0, 0]
>>> [e+1 for e in a if e<10]
[2, 3, 4]
>>> [print(i,e) for i,e in enumerate(a)]
0 1
1 10
2 2
3 3
[None, None, None, None]
```

Next time: sets, dictionaries, and tuples.

3 Exercises

1. You have a list with numbers. Compute the mean of this list.
2. You have a list of numbers. Create two lists: the list of positives and a list of negatives.
3. You have a list. Compute the sum of the first n-2 elements.
4. You have a list. Compute a list that contains all the elements from the odd positions.
5. You have a list of numbers. Find the minimum element of the list without using the min function.