Python Crash Course

This notebook will go through the basic topics:

- Data types
    - Numbers
    - Strings
    - Printing
    - Lists
    - Dictionaries
    - Booleans
    - Tuples
    - Sets
- Comparison Operators
- if, elif, else Statements
- for Loops
- while Loops
- range()
- list comprehension
- functions
- lambda expressions
- map and filter
- methods

Data types

Numbers

```
In [1]: 1 + 1
Out[1]: 2
```

```
In [2]: 1 * 3
Out[2]: 3
```

```
In [3]: 3 / 2
Out[3]: 1.5
```

```
In [4]: 3 // 2
Out[4]: 1
```

```
In [5]: 2 ** 4
Out[5]: 16
```

```
In [6]: 4 % 2
Out[6]: 0
```

```
In [7]: 5 % 2
Out[7]: 1
```

```
In [8]: (2 + 3) * (5 + 5)
Out[8]: 50
```

Comments

```
In [9]: # This is a single line comment

        """
        Tre quotes are used for multi-line strings,
        often used as multi-line comments or documentation
        """

        1 + 1  # we can add comments at the end of a line

Out[9]: 2
```

Variable Assignment

We can associate names to values to easily refer to them.

```
In [10]: # Can not start with number or special characters
         name_of_var = 2
```

```
In [11]: x = 2
         y = 3
```

```
In [12]: z = x + y
```

```
In [13]: z

Out[13]: 5
```

```
In [14]: googol = 10 ** 100
         googol

Out[14]: 1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         00000000000
```

```
In [15]: # python is dynamically typed, so this is possible
         x = 2
         x = 'hello'
```

Strings

- They can be placed inside apices (`'hello'`) or quotes (`"hello"`).
- The escape character is `\` , so you can do `'I\'m'` or `"I'm"` .

```
In [16]: 'single quotes'

Out[16]: 'single quotes'
```

```
In [17]: "double quotes"

Out[17]: 'double quotes'
```

```
In [18]: "wrap lot's of other quotes"

Out[18]: "wrap lot's of other quotes"
```

```
In [19]: # You can check the length of things with
         s = "hello"
         len(s)

Out[19]: 5
```

Printing

```
In [20]: x = 'hello'
```

```
In [21]: x
```

```
Out[21]: 'hello'
```

```
In [22]: print(x)
```

```
hello
```

```
In [23]: num = 12
         name = 'Sam'
```

```
In [24]: print('My number is: {one}, and my name is: {two}'.format(one=num, two=name))
```

```
My number is: 12, and my name is: Sam
```

```
In [25]: print('My number is: {}, and my name is: {}'.format(num, name))
```

```
My number is: 12, and my name is: Sam
```

```
In [26]: # For reference also a C-like syntax ia accepted
         print("My number is %d, and my name is %s" % (num, name))
```

```
My number is 12, and my name is Sam
```

Booleans

They can only be `True` or `False`

```
In [27]: True
```

```
Out[27]: True
```

```
In [28]: False
```

```
Out[28]: False
```

List

- A list contains an ordered sequence of elements
- A list is **mutable**: they can be altered (insertion, deletion, updates)

```
In [29]: [1, 2, 3]
```

```
Out[29]: [1, 2, 3]
```

```
In [30]: ['hi', 1, [1, 2]]
```

```
Out[30]: ['hi', 1, [1, 2]]
```

```
In [31]: my_list = ['a', 'b', 'c']
```

```
In [32]: my_list.append('d')
```

```
In [33]: my_list
```

```
Out[33]: ['a', 'b', 'c', 'd']
```

```
In [34]: my_list[0]
```

```
Out[34]: 'a'
```

```
In [35]: my_list[1]
Out[35]: 'b'
```

```
In [36]: my_list[-1]
Out[36]: 'd'
```

Slicing

```
      +---+---+---+---+---+---+
      | P | y | t | h | o | n |
      +---+---+---+---+---+---+
      0   1   2   3   4   5   6
     -6  -5  -4  -3  -2  -1
```

The syntax is `[from:to]`

```
In [37]: my_list[1:3]  # this is called slicing (from included, to excluded)
Out[37]: ['b', 'c']
```

```
In [38]: my_list[1:]   # if "to" is not provided, it's until the end
Out[38]: ['b', 'c', 'd']
```

```
In [39]: my_list[:2]   # if "from" is not provided, it's from the beginning
Out[39]: ['a', 'b']
```

```
In [40]: my_list[0] = 'NEW'  # list elements can be overrided
```

```
In [41]: my_list
Out[41]: ['NEW', 'b', 'c', 'd']
```

```
In [42]: nest = [1, 2, 3, [4, 5, ['target']]]
```

```
In [43]: nest[3]
Out[43]: [4, 5, ['target']]
```

```
In [44]: nest[3][2]
Out[44]: ['target']
```

```
In [45]: nest[3][2][0]
Out[45]: 'target'
```

Dictionaries

- They can associate **keys** to **values**
- They make it possible to easily retrieve the value by accessing the data using the key

```
In [46]: d = {'key1': 'item1', 'key2': 'item2'}
         d
Out[46]: {'key1': 'item1', 'key2': 'item2'}
```

```
In [47]: d['key1']
Out[47]: 'item1'
```

```
In [48]: d['key3'] = 'item3'
         d
Out[48]: {'key1': 'item1', 'key2': 'item2', 'key3': 'item3'}
```

Tuples

- A tuple is a group of elements
- Similar to a list, but it is **not mutable**

```
In [49]: t = (1, 2, 3)
```

```
In [50]: t[0]
Out[50]: 1
```

```
In [51]: t[0] = 'NEW'

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-51-93bfe9be1549> in <module>
         ----> 1 t[0] = 'NEW'

         TypeError: 'tuple' object does not support item assignment
```

Sets

- Unordered set of elements without duplicates.
- They support operations such as union, intersection, difference

```
In [52]: a = {1, 2, 3}
         a
Out[52]: {1, 2, 3}
```

```
In [53]: b = {2, 3, 4, 2, 4, 2, 3, 3, 3, 3, 2, 2, 2, 4, 5, 2}
         b
Out[53]: {2, 3, 4, 5}
```

```
In [54]: a.intersection(b)
Out[54]: {2, 3}
```

```
In [55]: a.union(b)
Out[55]: {1, 2, 3, 4, 5}
```

```
In [56]: a.difference(b)
Out[56]: {1}
```

When to use a list, a tuple, a set, or a dictionary?

- **list**: the order is preserved, they can contain duplicates, mutable
- **tuple**: the order is preserved, immutable
- **set**: no order, mutable, *inclusion checks are extremely fast*
- **dictionary**: no order, mutable, any type of keys

Comparison Operators

```
In [57]: 1 > 2
Out[57]: False
```

```
In [58]: 1 < 2.5
Out[58]: True
```

```
In [59]: 1 >= 1
Out[59]: True
```

```
In [60]: 1 <= 4
Out[60]: True
```

```
In [61]: 1 == 1
Out[61]: True
```

```
In [62]: 'hi' == 'bye'
Out[62]: False
```

```
In [63]: 'a' > 'b'
Out[63]: False
```

Logic Operators

```
In [64]: (1 > 2) and (2 < 3)
Out[64]: False
```

```
In [65]: (1 > 2) or (2 < 3)
Out[65]: True
```

```
In [66]: (1 == 2) or (2 == 3) or (4 == 4)
Out[66]: True
```

```
In [67]: not (2 < 3)
Out[67]: False
```

`if`, `elif`, `else` Statements

```
In [68]: if 1 < 2:
             print('Yep!')

         Yep!
```

```
In [69]: if 1 < 2:
             print('yep!')

         yep!
```

In [70]:
```
if 1 < 2:
    print('first')
else:
    print('last')
```
```
first
```

In [71]:
```
if 1 > 2:
    print('first')
else:
    print('last')
```
```
last
```

In [72]:
```
if 1 == 2:
    print('first')
elif 3 == 3:
    print('middle')
else:
    print('Last')
```
```
middle
```

for Loops

In [73]:
```
seq = [1, 2, 3, 4, 5]
```

In [74]:
```
for item in seq:
    print(item)
```
```
1
2
3
4
5
```

In [75]:
```
for item in seq:
    print('Yep')
```
```
Yep
Yep
Yep
Yep
Yep
```

In [76]:
```
for item in seq:
    print(item + item)
```
```
2
4
6
8
10
```

In [77]:
```
for item in range(10):
    if item % 2 == 0:
        continue   # terminates the execution for the current value
    if item % 7 == 0:
        break      # terminates the loop
    print(item)
```
```
1
3
5
```

while Loops

```
In [78]: i = 1
         while i < 5:
             print('i is: {}'.format(i))
             i = i + 1

         i is: 1
         i is: 2
         i is: 3
         i is: 4
```

range()

```
In [79]: range(5)

Out[79]: range(0, 5)
```

```
In [80]: for i in range(5):
             print(i)

         0
         1
         2
         3
         4
```

```
In [81]: list(range(5))

Out[81]: [0, 1, 2, 3, 4]
```

List comprehension

Python simplifies the generation of lists from other lists by avoiding the for loop.

```
In [82]: x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [83]: out = []
         for item in x:
             out.append(item ** 2)
         print(out)

         [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
In [84]: [item ** 2 for item in x]

Out[84]: [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
In [85]: [item ** 2 for item in x if item % 2 == 0]

Out[85]: [4, 16, 36, 64]
```

Functions

- Sequence of instruction
- zero or more inputs, one output

```
In [86]: def my_func(param1='default'):
             """
             Documentation goes here.
             """
             print(param1)
```

```
In [87]: my_func

Out[87]: <function __main__.my_func(param1='default')>
```

```
In [88]: my_func()

         default

In [89]: my_func('new param')

         new param

In [90]: my_func(param1='new param')

         new param

In [91]: def square(x):
             return x ** 2

In [92]: out = square(2)

In [93]: print(out)

         4
```

lambda expressions

```
In [94]: def times2(var):
             return var * 2

In [95]: times2(2)

Out[95]: 4

In [96]: lambda var: var*2

Out[96]: <function __main__.<lambda>(var)>
```

map and filter

```
In [97]: seq = [1, 2, 3, 4, 5]

In [98]: map(times2, seq)

Out[98]: <map at 0x10520d190>

In [99]: list(map(times2, seq))

Out[99]: [2, 4, 6, 8, 10]

In [100]: list(map(lambda var: var * 2, seq))

Out[100]: [2, 4, 6, 8, 10]

In [101]: filter(lambda item: item % 2 == 0, seq)

Out[101]: <filter at 0x10490f390>

In [102]: list(filter(lambda item: item % 2 == 0, seq))

Out[102]: [2, 4]
```

methods

```
In [103]: st = 'hello my name is Enrico'
In [104]: st.lower()

Out[104]: 'hello my name is enrico'

In [105]: st.upper()

Out[105]: 'HELLO MY NAME IS ENRICO'

In [106]: st.split()

Out[106]: ['hello', 'my', 'name', 'is', 'Enrico']

In [107]: tweet = 'Go Sports! #Sports'

In [108]: tweet.split('#')

Out[108]: ['Go Sports! ', 'Sports']

In [109]: tweet.split('#')[1]

Out[109]: 'Sports'

In [110]: d = {'a': 1, 'b': 2}

In [111]: d.keys()

Out[111]: dict_keys(['a', 'b'])

In [112]: d.items()

Out[112]: dict_items([('a', 1), ('b', 2)])

In [113]: lst = [1, 2, 3]

In [114]: lst.pop()

Out[114]: 3

In [115]: lst

Out[115]: [1, 2]

In [116]: 'x' in [1, 2, 3]

Out[116]: False

In [117]: 'x' in ['x', 'y', 'z']

Out[117]: True

In [118]: 'x' in 'abcxyz'

Out[118]: True
```