Introduction to Python: Part 1

Basics

Python as a calculator

In [7]:
```
1 + 12
```

Out[7]: 13

In [8]:
```
24 - 3 * 4
```

Out[8]: 12

In [9]:
```
(24 - 3) * 4
```

Out[9]: 84

In [10]:
```
4 / 2
```

Out[10]: 2.0

In [11]:
```
2 ** 4 # 2 to the power of 4
```

Out[11]: 16

In [12]:
```
14 > 10
```

Out[12]: True

In [13]:
```
12 + 15 == 27
```

Out[13]: True

In [14]:
```
14 % 6 # remainder of 14 divided by 6
```

Out[14]: 2

The print() function

The basic way to output is the print function

In [15]:
```
print('Hello world')
```

```
Hello world
```

In [16]:
```
print(1 + 2)
print(7 - 1)
```

```
3
6
```

Atomic data types

- Boolean values (True, False)
- Integers (3, 4, 8)
- Floating-point numbers (3.1416, 5.8274)
- Letters and strings ('a', 'pqm')

Creating an object of a particular data type

In [17]:
```
# Creates a boolean object
has_insurance = True

# Creates an integer object
i = 24

# Creates a floating-point number object
cash = 28.35

# Creates a string object
sport = 'soccer'
```

Display the data type using the function type()

In [18]:
```
print(type(has_insurance))
print(type(i))
print(type(cash))
print(type(sport))
```

```
<class 'bool'>
<class 'int'>
<class 'float'>
<class 'str'>
```

Display string and integer objects simultaneously using the print() function

In [19]:
```
my_number = 10
print('My number is ' + str(10))
```

```
My number is 10
```

Built-in data structures

- Tuple: ordered sequence of objects of any type that cannot be changed (immutable)
- List: ordered sequence of objects of any type that can be changed (mutable)
- Dictionary: unordered collection of objects of any type

Tuple

- Every element rests at some position (index) in the tuple

- The index can be used to locate a particular element
- The first index begins at zero, the next is one, ect.
- Tuples are immutable, i.e., they cannot be modified

Tuples can be created by using the ()-brackets

In [20]:
```
new_product = ('Apple', 'iPhone XS', 515, 1639, 'Gold')
new_product
```

Out[20]:
```
('Apple', 'iPhone XS', 515, 1639, 'Gold')
```

Values of tuples can be accessed directly

In [21]:
```
new_product[0]
```

Out[21]:
```
'Apple'
```

In [22]:
```
new_product[0:2]
```

Out[22]:
```
('Apple', 'iPhone XS')
```

In [23]:
```
new_product[:3]
```

Out[23]:
```
('Apple', 'iPhone XS', 515)
```

In [24]:
```
new_product[-1]
```

Out[24]:
```
'Gold'
```

In [25]:
```
new_product[-3:]
```

Out[25]:
```
(515, 1639, 'Gold')
```

Tuples are immutable, i.e., they cannot be changed

In [26]:
```
new_product[0] = 'Samsung'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-26-3a96041cf17d> in <module>()
----> 1 new_product[0] = 'Samsung'

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

List

- Every element rests at some position (index) in the list
- The index can be used to locate a particular element
- The first index begins at zero, the next is one, ect.

Lists can be created by using the []-brackets

In [27]:
```
stocks = ['ABB', 'UBS', 'ATL']
stocks
```

Out[27]:
```
['ABB', 'UBS', 'ATL']
```

Lists can be created by using the functions list() and range()

In [28]:
```
x = list(range(10))
x
```

Out[28]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lists can contain any type of objects, including other lists

In [29]:
```
housing = ['own', 'rent']
assortment = ['Analytics', 7.5, housing]
assortment
```

Out[29]:
```
['Analytics', 7.5, ['own', 'rent']]
```

Values of lists can be accessed directly using an index

In [30]:
```
stocks[0]
```

Out[30]:
```
'ABB'
```

In [31]:
```
stocks[1]
```

Out[31]:
```
'UBS'
```

In [32]:
```
stocks[:2]
```

Out[32]:
```
['ABB', 'UBS']
```

In [33]:
```
stocks[-1]
```

Out[33]:
```
'ATL'
```

In [34]:
```
stocks[-2:]
```

Out[34]:
```
['UBS', 'ATL']
```

In [35]:
```
stocks[1:]
```

Out[35]:
```
['UBS', 'ATL']
```

Values can easily be appended to lists

In [36]:
```
stocks.append('NOVN')
stocks
```

Out[36]:
```
['ABB', 'UBS', 'ATL', 'NOVN']
```

Values can easily be deleted from lists

In [37]:
```
del x[-1]
x
```

Out[37]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [38]:
```
del x[1:3]
x
```

Out[38]:
```
[0, 3, 4, 5, 6, 7, 8]
```

Values of lists can be changed

In [39]:
```
x[0] = 3
x
```

Out[39]:
```
[3, 3, 4, 5, 6, 7, 8]
```

Dictionary

- Every element is associated with a key
- The key can be used to retrieve an object

Dictionaries can be created by using the {}-brackets

In [40]:
```
house_attributes = {'num_rooms': 4,
                    'space_square_m': 120,
                    'balcony': True}
house_attributes
```

Out[40]:
```
{'balcony': True, 'num_rooms': 4, 'space_square_m': 120}
```

Values can be accessed using the key

In [41]:
```
house_attributes['balcony']
```

Out[41]:
```
True
```

Elements can easily be added

In [42]:
```
house_attributes['num_bathrooms'] = 2
house_attributes
```

Out[42]:
```
{'balcony': True, 'num_bathrooms': 2, 'num_rooms': 4, 'space_square_m': 120}
```

Values can be updated

In [43]:
```
house_attributes['space_square_m'] = 130
house_attributes
```

Out[43]:
```
{'balcony': True, 'num_bathrooms': 2, 'num_rooms': 4, 'space_square_m': 130}
```

Various **immutable** objects can be used as keys

In [44]:
```
my_dict = {(1, 2): 10,
           'color': 'red',
           5: [1, 2, 3, 4, 5]}
my_dict
```

Out[44]:
```
{(1, 2): 10, 'color': 'red', 5: [1, 2, 3, 4, 5]}
```

In [45]:
```
my_dict = {[1, 2]: 10}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-45-c9ae3a4ad8e7> in <module>()
----> 1 my_dict = {[1, 2]: 10}

TypeError: unhashable type: 'list'
```

Conditionals and loops

- if-statement
- for-statement

if-statement

- Evaluates a logical condition
- If the condition is true, then the indented statements are executed
- If the condition is not true, then the indented statements are skipped.

In [46]:
```
amount = 150
if amount > 100:
    print('You purchased for more than 100 CHF. We can offer you a discount.')
```

```
You purchased for more than 100 CHF. We can offer you a discount.
```

for-statement

- Iterates over elements of a sequence in order
- The indented statements are executed for each element

Can be used to print each element of a list

In [47]:
```
prices = [14, 15, 26, 33, 75]
for price in prices:
    print('The price is ' + str(price) + ' CHF')
```

```
The price is 14 CHF
The price is 15 CHF
The price is 26 CHF
The price is 33 CHF
The price is 75 CHF
```

Can be used to modify elements of a list

In [48]:
```
for i in [0, 1, 4]:
    prices[i] = prices[i] + 1
prices
```

Out[48]:
```
[15, 16, 26, 33, 76]
```

In [ ]:
```

```