

1.1-Python Intro

[KeytoDataScience.com](https://keytoDataScience.com)

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

1. web development (server-side),
2. software development,
3. mathematics,
4. system scripting.

What can Python do?

1. Python can be used on a server to create web applications.
2. Python can be used alongside software to create workflows.
3. Python can connect to database systems. It can also read and modify files.
4. Python can be used to handle big data and perform complex mathematics.
5. It can be used for machine learning applications, data analytics, etc.,

Why Python?

1. Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
2. Python has a simple syntax similar to the English language.
3. Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
4. Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
5. Python can be treated in a procedural way, an object-orientated way or a functional way.

Good to know
















The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

Python Syntax compared to other programming languages

1. Python was designed for readability, and has some similarities to the English language with influence from mathematics.
2. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
3. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Getting started with Python

This notebook will just go through the basic topics in order:

1. Python Indentation 
 2. Python comments 
 3. Python variables 
 4. Data types 
 - 4.1. Built-in Data Types
 - 4.2. Getting the Data Type
 - 4.3. Setting the Data Type
 - 4.4. Setting the Specific Data Type
 - 4.5. Numbers
 - 4.6. Casting: Specifying a Variable Type
 - 4.7. Strings
 - 4.8. Escape Character
 - 4.9. Lists
 - 4.10. Tuples
 - 4.11. Dictionaries
 - 4.12. Nesting with Dictionary
 - 4.13. Dictionary Methods
 - 4.14. Booleans
 - 4.15. Sets
 5. Comparison Operators 
 6. Logic Operators 
 7. if, elif, else Statements 
 8. for Loops 
 9. while Loops 
 10. range() 
 11. List Comprehension 
 12. Functions 
 13. Lambda Expressions 
 14. Map and Filter 
 15. Refresher 
-

1 Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

```
In [1]: import sys
        print(sys.version)
```

3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)]

```
In [2]: if 5 > 2:
        print("Five is greater than two!")
```

Five is greater than two!

```
In [3]: #Python will give you an error if you skip the indentation:

        if 5 > 2:
        print("Five is greater than two!")
```

```
File "C:\Users\Prateek\AppData\Local\Temp\ipykernel_8408\35213069.py", line 4
    print("Five is greater than two!")
    ^
```

IndentationError: expected an indented block

```
In [4]: #The number of spaces is up to you as a programmer, but it has to be at least one.

        if 5 > 2:
        print("Five is greater than two!")
        if 5 > 2:
            print("Five is greater than two!")
```

Five is greater than two!
Five is greater than two!

```
In [2]: #You have to use the same number of spaces in the same block of code, otherwise Python

        if 5 > 2:
        print("Five is greater than two!")
            print("Five is greater than two!")
```

[↑ back to top](#)

2 Python Comments

Comments can be used to explain Python code. Comments can be used to make the code more readable. Comments can be used to prevent execution when testing code.

Creating a Comment Comments starts with a #, and Python will ignore them:

In [147...

```
#This is a comment
print("Hello, World!")

#Comments can be placed at the end of a line, and Python will ignore the rest of the Li
print("Hello, World!") #This is a comment

#Comments does not have to be text to explain the code, it can also be used to prevent
#print("Hello, World!")
print("Cheers, Mate!")
```

```
Hello, World!
Hello, World!
Cheers, Mate!
```

2.1 Multi Line Comments

Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

In [6]:

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
Hello, World!
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

In [7]:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
Hello, World!
```

[↑ back to top](#)

3 Python Variables

3.1 Creating Variables

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

In [8]:

```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

In [9]:

```
#Variables do not need to be declared with any particular type and can even change type
```

```
x = 4 # x is of type int
x = "Sally" # x is now of type str
print(x)
```

```
Sally
```

3.2 Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

1. A variable name must start with a letter or the underscore character
2. A variable name cannot start with a number
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
4. Variable names are case-sensitive (age, Age and AGE are three different variables)

Example:

- Legal variable names:
 - myvar = "John"
 - my_var = "John"
 - _my_var = "John"
 - myVar = "John"
 - MYVAR = "John"
 - myvar2 = "John"
- Illegal variable names:
 - 2myvar = "John"
 - my-var = "John"
 - my var = "John"

In [6]:

```
"""
Assign Value to Multiple Variables
Python allows you to assign values to multiple variables in one line:
"""
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
"""
And you can assign the same value to multiple variables in one line:
"""
a = b = c = "Orange"
print(a)
print(b)
print(c)
```

Orange
Banana
Cherry
Orange
Orange
Orange

In [11]:

```
#The Python print statement is often used to output variables.
#To combine both text and a variable, Python uses the + character:
x = "awesome"
print("Python is " + x)

#You can also use the + character to add a variable to another variable:
x = "Python is "
y = "awesome"
z = x + y
print(z)

#For numbers, the + character works as a mathematical operator:
x = 5
y = 10
print(x + y)

#If you try to combine a string and a number, Python will give you an error:
x = 5
y = "John"
print(x + y)
```

Python is awesome
Python is awesome
15

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8408\2559575861.py in <module>
    18 x = 5
    19 y = "John"
--> 20 print(x + y)
```

TypeError: unsupported operand type(s) for +: 'int' and 'str'

[↑ back to top](#)

4 Data types

4.1 Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type: str Numeric Types: int, float, complex Sequence Types: list, tuple, range Mapping Type: dict Set Types: set, frozenset Boolean Type: bool Binary Types: bytes, bytearray, memoryview

4.2 Getting the Data Type

You can get the data type of any object by using the `type()` function

```
In [12]: x = 5
         print(type(x))
```

```
<class 'int'>
```

4.3 Setting the Data Type

```
In [13]: var1 = "Hello World"      #str
         var2 = 20                  #int
         var3 = 20.5                #float
         var4 = 1j                  #complex
         var5 = ["apple", "banana", "cherry"]  #list
         var6 = ("apple", "banana", "cherry")  #tuple
         var7 = range(6)            #range
         var8 = {"name" : "John", "age" : 36}   #dict
         var9 = {"apple", "banana", "cherry"}  #set
         var10 = frozenset({"apple", "banana", "cherry"}) #frozenset
         var11 = True                #bool
         var12 = b"Hello"            #bytes
         var13 = bytearray(5)        #bytearray
         var14 = memoryview(bytes(5)) #memoryview
```

```
In [14]: print(var1,type(var1))
         print(var2,type(var2))
         print(var3,type(var3))
         print(var4,type(var4))
         print(var5,type(var5))
         print(var6,type(var6))
         print(var7,type(var7))
         print(var8,type(var8))
         print(var9,type(var9))
         print(var10,type(var10))
         print(var11,type(var11))
         print(var12,type(var12))
         print(var13,type(var13))
         print(var14,type(var14))
```

```
Hello World <class 'str'>
20 <class 'int'>
20.5 <class 'float'>
1j <class 'complex'>
['apple', 'banana', 'cherry'] <class 'list'>
('apple', 'banana', 'cherry') <class 'tuple'>
range(0, 6) <class 'range'>
{'name': 'John', 'age': 36} <class 'dict'>
```

```
{'banana', 'cherry', 'apple'} <class 'set'>
frozenset({'banana', 'cherry', 'apple'}) <class 'frozenset'>
True <class 'bool'>
b'Hello' <class 'bytes'>
bytearray(b'\x00\x00\x00\x00\x00') <class 'bytearray'>
<memory at 0x000001C282A93A00> <class 'memoryview'>
```

4.4 Setting the Specific Data Type

```
In [15]: var1 = str("Hello World")      #str
var2 = int(20)                        #int
var3 = float(20.5)                   #float
var4 = complex(1j)                   #complex
var5 = list(("apple", "banana", "cherry")) #list
var6 = tuple(("apple", "banana", "cherry")) #tuple
var7 = range(6)                      #range
var8 = dict(name="John", age= 36)     #dict
var9 = set(("apple", "banana", "cherry")) #set
var10 = frozenset(("apple", "banana", "cherry")) #frozenset
var11 = bool(5)                      #bool
var12 = bytes(5)                    #bytes
var13 = bytearray(5)                #bytearray
var14 = memoryview(bytes(5))         #memoryview
```

```
In [16]: print(var1,type(var1))
print(var2,type(var2))
print(var3,type(var3))
print(var4,type(var4))
print(var5,type(var5))
print(var6,type(var6))
print(var7,type(var7))
print(var8,type(var8))
print(var9,type(var9))
print(var10,type(var10))
print(var11,type(var11))
print(var12,type(var12))
print(var13,type(var13))
print(var14,type(var14))
```

```
Hello World <class 'str'>
20 <class 'int'>
20.5 <class 'float'>
1j <class 'complex'>
['apple', 'banana', 'cherry'] <class 'list'>
('apple', 'banana', 'cherry') <class 'tuple'>
range(0, 6) <class 'range'>
{'name': 'John', 'age': 36} <class 'dict'>
{'banana', 'cherry', 'apple'} <class 'set'>
frozenset({'banana', 'cherry', 'apple'}) <class 'frozenset'>
True <class 'bool'>
b'\x00\x00\x00\x00\x00' <class 'bytes'>
bytearray(b'\x00\x00\x00\x00\x00') <class 'bytearray'>
<memory at 0x000001C282A93AC0> <class 'memoryview'>
```

4.5 Numbers

There are three numeric types in Python:

1. int
2. float
3. complex

Variables of numeric types are created when you assign a value to them

```
In [17]: #Int: Int, or integer, is a whole number, positive or negative, without decimals, of un
x = 1
y = 35656222554887711
z = -3255522

print(x,type(x))
print(y,type(y))
print(z,type(z))
```

```
1 <class 'int'>
35656222554887711 <class 'int'>
-3255522 <class 'int'>
```

```
In [18]: #Float: Float, or "floating point number" is a number, positive or negative, containing
x = 1.10
y = 1.0
z = -35.59

print(x,type(x))
print(y,type(y))
print(z,type(z))
```

```
1.1 <class 'float'>
1.0 <class 'float'>
-35.59 <class 'float'>
```

```
In [19]: #Float can also be scientific numbers with an "e" to indicate the power of 10.

x = 35e3
y = 12E4
z = -87.7e100

print(x,type(x))
print(y,type(y))
print(z,type(z))
```

```
35000.0 <class 'float'>
120000.0 <class 'float'>
-8.77e+101 <class 'float'>
```

```
In [20]: #Complex: Complex numbers are written with a "j" as the imaginary part:

x = 3+5j
y = 5j
z = -5j

print(x,type(x))
print(y,type(y))
print(z,type(z))
```

```
(3+5j) <class 'complex'>
5j <class 'complex'>
(-0-5j) <class 'complex'>
```

4.6 Casting: Specifying a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number) float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer) str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
In [21]: #Integers
x = int(1)    # x will be 1
y = int(2.8)  # y will be 2
z = int("3")  # z will be 3
w = int(True) # w will be 1
```

```
In [22]: #Floats
x = float(1)    # x will be 1.0
y = float(2.8)  # y will be 2.8
z = float("3")  # z will be 3.0
w = float("4.2") # w will be 4.2
```

```
In [23]: #Strings
x = str("s1")  # x will be 's1'
y = str(2)     # y will be '2'
z = str(3.0)   # z will be '3.0'
```

4.7 Strings

```
In [24]: 'single quotes'
```

```
Out[24]: 'single quotes'
```

```
In [25]: "double quotes"
```

```
Out[25]: 'double quotes'
```

```
In [26]: " wrap lot's of other quotes"
```

```
Out[26]: " wrap lot's of other quotes"
```

In [27]:

```
print("""this is a  
multiline  
string""")
```

```
this is a  
multiline  
string
```

In [28]:

```
#Strings are Arrays  
string = "Hello, World!"  
print(string[1])
```

```
e
```

In [29]:

```
#Slicing  
print(string[2:5])  
print(string[:])  
print(string[-5:-2])
```

```
llo  
Hello, World!  
orl
```

In [30]:

```
#Finding the length of the string  
print(len(string))
```

```
13
```

String Methods

Python has a set of built-in methods that you can use on strings.

- The strip() method removes any whitespace from the beginning or the end:
- The upper() method returns the string in upper case:
- The replace() method replaces a string with another string:
- The split() method splits the string into substrings if it finds instances of the separator:

Check words in a String

To check if a certain phrase or character is present in a string, we can use the keywords in or not in.

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

String Format

- As we learned in the Python Variables chapter, we cannot combine strings and numbers.
- But we can combine strings and numbers by using the format() method!
- The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are

- The format() method takes unlimited number of arguments, and are placed into the respective placeholders
- You can use index numbers {0} to be sure the arguments are placed in the correct placeholders

In [31]:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"

a = "Hello, World!"
print(a.lower()) # returns "hello, world!"
print(a.upper()) # returns "HELLO, WORLD!"

print(a.replace("H", "J")) # returns Jello, World!
print(a.split(",")) # returns ['Hello', ' World!']

txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x) # returns True

x = "ain" not in txt
print(x) # returns False

a = "Hello"
b = "World"
c = a + b
print(c) # returns HelloWorld

a = "Hello"
b = "World"
c = a + " " + b
print(c) # returns Hello World

age = 36
txt = "My name is John, and I am {}"
print(txt.format(age)) # returns My name is John, and I am 36

quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price)) # returns I want 3 pieces of item 567 fo

quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price)) # returns I want to pay 49.95 dollars fo

num = 12
name = 'Sam'
print('My number is: {one}, and my name is: {two}'.format(one=num,two=name)) # returns .
```

```
Hello, World!
hello, world!
HELLO, WORLD!
Jello, World!
['Hello', ' World!']
```

```
True
False
HelloWorld
Hello World
My name is John, and I am 36
I want 3 pieces of item 567 for 49.95 dollars.
I want to pay 49.95 dollars for 3 pieces of item 567.
My number is: 12, and my name is: Sam
```

4.8 Escape Character

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes
- The escape character allows you to use double quotes when you normally would not be allowed

```
In [32]: txt = "We are the so-called "Vikings" from the north."
```

```
File "C:\Users\Prateek\AppData\Local\Temp\ipykernel_8408\7934146.py", line 1
    txt = "We are the so-called "Vikings" from the north."
                                   ^
```

SyntaxError: invalid syntax

```
In [33]: txt = "We are the so-called \"Vikings\" from the north."
        print(txt)
```

We are the so-called "Vikings" from the north.

4.9 Lists

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

```
In [12]: thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
        #Access Items
        print(thislist[1])
        #Negative Indexing
        print(thislist[-1])
        #Range of Indexes
        print(thislist[2:5])
```

```
banana
mango
['cherry', 'orange', 'kiwi']
```

```
In [35]: print(thislist[:4])
        print(thislist[2:])
        print(thislist[-4:-1])
```

```
['apple', 'banana', 'cherry', 'orange']
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

```
['orange', 'kiwi', 'melon']
```

```
In [36]: nest = [1,2,3,[4,5,['target']]]
```

```
In [37]: print(nest[3])
print(nest[3][2])
print(nest[3][2][0])
```

```
[4, 5, ['target']]
['target']
target
```

```
In [38]: thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

```
In [39]: if "apple" in thislist:
print("Yes, 'apple' is in the fruits list")
```

```
Yes, 'apple' is in the fruits list
```

```
In [40]: print(thislist)
print(len(thislist))
```

```
['apple', 'blackcurrant', 'cherry']
3
```

```
In [41]: thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

```
In [42]: thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

```
In [43]: thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry']
```

```
In [44]: thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

```
['apple', 'banana']
```

```
In [45]: thislist = ["apple", "banana", "cherry"]
del thislist
print(thislist)
```

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8408\3386762020.py in <module>
      1 thislist = ["apple", "banana", "cherry"]
      2 del thislist
----> 3 print(thislist)

NameError: name 'thislist' is not defined
```

```
In [46]: thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

```
In [47]: thislist = ["apple", "banana", "cherry"]
mylist = thislist
mylist[0] = "orange"
print(mylist)
print(thislist)
```

```
['orange', 'banana', 'cherry']
['orange', 'banana', 'cherry']
```

```
In [48]: thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
mylist[0] = "orange"
print(mylist)
print(thislist)
```

```
['orange', 'banana', 'cherry']
['apple', 'banana', 'cherry']
```

```
In [49]: list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
In [50]: list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

for x in list2:
    list1.append(x)

print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
In [51]: list1 = ["a", "b", "c"]
```

```
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

4.10 Tuples

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

```
In [66]: t = ("apple", "banana", "cherry")
```

```
In [67]: t[0]
```

```
Out[67]: 'apple'
```

```
In [68]: t[0] = 'NEW'
```

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8408\2140988817.py in <module>
----> 1 t[0] = 'NEW'
```

TypeError: 'tuple' object does not support item assignment

```
In [69]: x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x, type(x))
```

```
('apple', 'kiwi', 'cherry') <class 'tuple'>
```

```
In [70]: #Create Tuple With One Item
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

```
In [71]: #Tuples are unchangeable, so you cannot remove items from it, but you can delete the tu
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```



```

NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8408\391015210.py in <module>
      2 thistuple = ("apple", "banana", "cherry")
      3 del thistuple
----> 4 print(thistuple) #this will raise an error because the tuple no longer exists

NameError: name 'thistuple' is not defined

```

```

In [72]: #Joining two tuples
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)

('a', 'b', 'c', 1, 2, 3)

```

4.11 Dictionaries

Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.

Constructing a Dictionary Let's see how we can construct dictionaries to get a better understanding of how they work!

```

In [52]: # Make a dictionary with {} and : to signify a key and a value
my_dict = {'key1':'value1','key2':'value2'}

```

```

In [53]: # Call values by their key
my_dict['key2']

```

```

Out[53]: 'value2'

```

```

In [54]: #Its important to note that dictionaries are very flexible in the data types they can h
my_dict = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}
print(my_dict)

{'key1': 123, 'key2': [12, 23, 33], 'key3': ['item0', 'item1', 'item2']}

```

```

In [55]: #Lets call items from the dictionary
print(my_dict['key3'])

# Can call an index on that value
print(my_dict['key3'][0])

#Can then even call methods on that value
print(my_dict['key3'][0].upper())

['item0', 'item1', 'item2']
item0
ITEM0

```

```

In [56]: #We can effect the values of a key as well. For instance:
print(my_dict['key1'])

```

```
# Subtract 123 from the value
my_dict['key1'] = my_dict['key1'] - 123

#Check
print(my_dict['key1'])
```

```
123
0
```

```
In [57]: """
A quick note, Python has a built-in method of doing a self subtraction or addition (or
We could have also used += or -= for the above statement. For example:
"""

# Set the object equal to itself minus 123
my_dict['key1'] -= 123
my_dict['key1']
```

```
Out[57]: -123
```

```
In [58]: #We can also create keys by assignment. For instance if we started off with an empty di

# Create a new dictionary
d = {}

# Create a new key through assignment
d['animal'] = 'Dog'

# Can do this with any object
d['answer'] = 42

#Show
d
```

```
Out[58]: {'animal': 'Dog', 'answer': 42}
```

4.12 Nesting with Dictionaries

Hopefully you're starting to see how powerful Python is with its flexibility of nesting objects and calling methods on them. Let's see a dictionary nested inside a dictionary:

```
In [59]: # Dictionary nested inside a dictionary nested in side a dictionary
d = {'key1':{'nestkey':{'subnestkey':'value'}}}

#Wow! That's a quite the inception of dictionaries! Let's see how we can grab that valu

# Keep calling the keys
d['key1']['nestkey']['subnestkey']
```

```
Out[59]: 'value'
```

4.13 A few Dictionary Methods

There are a few methods we can call on a dictionary. Let's get a quick introduction to a few of them:

In [60]:

```
# Create a typical dictionary
d = {'key1':1,'key2':2,'key3':3}

# Method to return a list of all keys
print(d.keys())

# Method to grab all values
print(d.values())

# Method to return tuples of all items (we'll learn about tuples soon)
print(d.items())
```

```
dict_keys(['key1', 'key2', 'key3'])
dict_values([1, 2, 3])
dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```

In [61]:

```
for key,value in d.items():
    print("Key: ",key,"Value: ",value)
```

```
Key:  key1 ,Value:  1
Key:  key2 ,Value:  2
Key:  key3 ,Value:  3
```

4.14 Booleans

Booleans represent one of two values: True or False.

Boolean Values

- In programming you often need to know if an expression is True or False.
- You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

In [62]:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

In [63]:

```
a = 200
b = 33

if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

```
b is not greater than a
```

Most Values are True

- Almost any value is evaluated to True if it has some sort of content.
- Any string is True, except empty strings.
- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.

In [64]:

```
#True
print(bool("abc"))
print(bool(123))
print(bool(["apple", "cherry", "banana"]))
```

True
True
True

In [65]:

```
#False
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

False
False
False
False
False
False
False

4.15 Sets

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

In [73]:

```
{1,2,3}
```

Out[73]: {1, 2, 3}

In [74]:

```
{1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2}
```

Out[74]: {1, 2, 3}

In [75]:

```
#Add an item to a set, using the add() method:

thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

{'banana', 'orange', 'cherry', 'apple'}

In [76]: *#Add multiple items to a set, using the update() method:*

```
thisset = {"apple", "banana", "cherry"}
thisset.update(["orange", "mango", "grapes"])
print(thisset)
```

{'banana', 'orange', 'grapes', 'cherry', 'mango', 'apple'}

In [77]: `thisset[0]`

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8408\235757851.py in <module>
----> 1 thisset[0]
```

TypeError: 'set' object is not subscriptable

In [78]: *#Remove: Remove "banana" by using the remove() method:*

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

```
thisset.remove("banana")
print(thisset)
```

{'cherry', 'apple'}

```
-----
KeyError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8408\3760388499.py in <module>
      5 print(thisset)
      6
----> 7 thisset.remove("banana")
      8 print(thisset)
```

KeyError: 'banana'

In [79]: *#Discard: Remove "banana" by using the discard() method:*

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
thisset.discard("banana")
print(thisset)
```

{'cherry', 'apple'}

{'cherry', 'apple'}

In [80]: *#Join Two Sets:*

#Union: The union() method returns a new set with all items from both sets:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
```

```
set3 = set1.union(set2)
print(set3)
```

#Update: The update() method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set1.update(set2)  
print(set1)
```

```
{1, 'a', 'b', 2, 3, 'c'}  
{1, 'a', 'b', 2, 3, 'c'}
```

[↑ back to top](#)

5 Comparison Operators

Comparison operators are used to compare two values:

```
In [81]: 1 > 2
```

```
Out[81]: False
```

```
In [82]: 1 < 2
```

```
Out[82]: True
```

```
In [83]: 1 >= 1
```

```
Out[83]: True
```

```
In [84]: 1 <= 4
```

```
Out[84]: True
```

```
In [85]: 1 == 1
```

```
Out[85]: True
```

```
In [86]: 'hi' == 'bye'
```

```
Out[86]: False
```

[↑ back to top](#)

6 Logic Operators

Logical operators are used to combine conditional statements:

```
In [87]: (1 > 2) and (2 < 3)
```

Out[87]: False

In [88]: `(1 > 2) or (2 < 3)`

Out[88]: True

In [89]: `(1 == 2) or (2 == 3) or (4 == 4)`

Out[89]: True

In [90]: `not(3 < 5 and 2 < 10)`

Out[90]: False

[↑ back to top](#)

7 if, elif, else Statements

In [91]: `if 1 < 2:
 print('Yep!')`

Yep!

In [13]: `if 1 > 2:
 print('yep!')`

In [93]: `if 1 < 2:
 print('first')
else:
 print('last')`

first

In [94]: `if 1 > 2:
 print('first')
else:
 print('last')`

last

In [95]: `if 1 == 2:
 print('first')
elif 3 == 3:
 print('middle')
else:
 print('Last')`

middle

In [96]:

```
x = 41

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        pass
```

Above ten,
and also above 20!

[↑ back to top](#)

8 for Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
In [15]: seq = [1,2,3,4,5]
```

```
In [98]: for item in seq:
          print(item)
```

1
2
3
4
5

```
In [99]: for item in 'python':
          print(item)
```

p
y
t
h
o
n

```
In [17]: for jelly in seq:
          print(jelly+jelly)
```

1
4
9
16
25

9 while Loops

With the while loop we can execute a set of statements as long as a condition is true.

In [101...

```
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
```

In [102...

*#With the break statement we can stop the loop even if the while condition is true:
#Exit the loop when i is 3:*

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

```
1
2
3
```

In [103...

*#With the continue statement we can stop the current iteration, and continue with the n
#Continue to the next iteration if i is 3:*

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
1
2
4
5
6
```

10 range()

In [104...

```
range(5)
```

Out[104...

```
range(0, 5)
```

```
In [105... for i in range(5):  
            print(i)
```

```
0  
1  
2  
3  
4
```

```
In [106... list(range(5))
```

```
Out[106... [0, 1, 2, 3, 4]
```

[↑ back to top](#)

11 list comprehension

```
In [107... x = [1,2,3,4]
```

```
In [108... out = []  
for item in x:  
    out.append(item**2)  
print(out)
```

```
[1, 4, 9, 16]
```

```
In [109... [item**2 for item in x]
```

```
Out[109... [1, 4, 9, 16]
```

```
In [110... # Check for even numbers in a range  
lst = [x for x in range(11) if x % 2 == 0]  
print(lst)
```

```
[0, 2, 4, 6, 8, 10]
```

```
In [111... # Convert Celsius to Fahrenheit  
celsius = [0,10,20.1,34.5]  
  
fahrenheit = [ ((float(9)/5)*temp + 32) for temp in celsius ]  
  
fahrenheit
```

```
Out[111... [32.0, 50.0, 68.18, 94.1]
```

```
In [112... # We can also perform nested list comprehensions, for example:  
lst = [ x**2 for x in [x**2 for x in range(11)]]  
lst
```

```
Out[112... [0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561, 10000]
```

12 Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a Function

- In Python a function is defined using the def keyword

```
In [113... def my_func(param1='default'):  
    """  
    Docstring goes here.  
    """  
    print(param1)
```

```
In [114... print(my_func.__doc__)  
print(my_func)
```

Docstring goes here.

<function my_func at 0x000001C282AC8160>

```
In [115... my_func()
```

default

```
In [116... my_func('new param')
```

new param

```
In [117... my_func(param1='new param')
```

new param

```
In [118... def square(x):  
    return x**2
```

```
In [119... out = square(2)
```

```
In [120... print(out)
```

4

13 Lambda Expressions

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

```
In [121...  
def times2(var):  
    return var*2
```

```
In [122...  
times2(2)
```

```
Out[122... 4
```

```
In [123...  
lambda var: var*2
```

```
Out[123... <function __main__.<lambda>(var)>
```

[↑ back to top](#)

14 Map and Filter

```
In [124...  
seq = [1,2,3,4,5]
```

```
In [125...  
map(times2,seq)
```

```
Out[125... <map at 0x1c2829c8dc0>
```

```
In [126...  
list(map(times2,seq))
```

```
Out[126... [2, 4, 6, 8, 10]
```

```
In [127...  
list(map(lambda var: var*2,seq))
```

```
Out[127... [2, 4, 6, 8, 10]
```

```
In [128...  
filter(lambda item: item%2 == 0,seq)
```

```
Out[128... <filter at 0x1c2829c8880>
```

```
In [129...  
list(filter(lambda item: item%2 == 0,seq))
```

```
Out[129... [2, 4]
```

[↑ back to top](#)

15 Refresher

```
In [130...
```

```
st = 'hello my name is Sam'
```

```
In [131... st.lower()
```

```
Out[131... 'hello my name is sam'
```

```
In [132... st.upper()
```

```
Out[132... 'HELLO MY NAME IS SAM'
```

```
In [133... st.split()
```

```
Out[133... ['hello', 'my', 'name', 'is', 'Sam']
```

```
In [134... tweet = 'Go Sports! #Sports'
```

```
In [135... tweet.split('#')
```

```
Out[135... ['Go Sports! ', 'Sports']
```

```
In [136... tweet.split('#')[1]
```

```
Out[136... 'Sports'
```

```
In [137... d = {'key1':1, 'key2':2, 'key3':3}
```

```
In [138... d.keys()
```

```
Out[138... dict_keys(['key1', 'key2', 'key3'])
```

```
In [139... d.items()
```

```
Out[139... dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```

```
In [140... lst = [1,2,3]
```

```
In [141... lst.pop()
```

```
Out[141... 3
```

```
In [142... lst
```

```
[1, 2]
```

Out[142...

In [143... `'x' in [1,2,3]`

Out[143... **False**

In [144... `'x' in ['x','y','z']`

Out[144... **True**

[↑ back to top](#)

Great Job!

[KeytoDataScience.com](https://keytoDataScience.com)