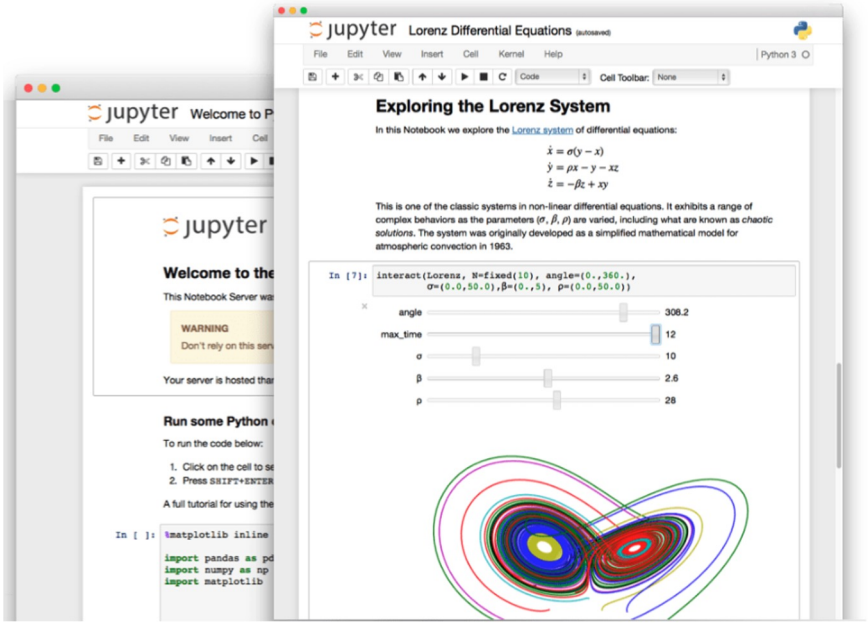
Introduction to Python

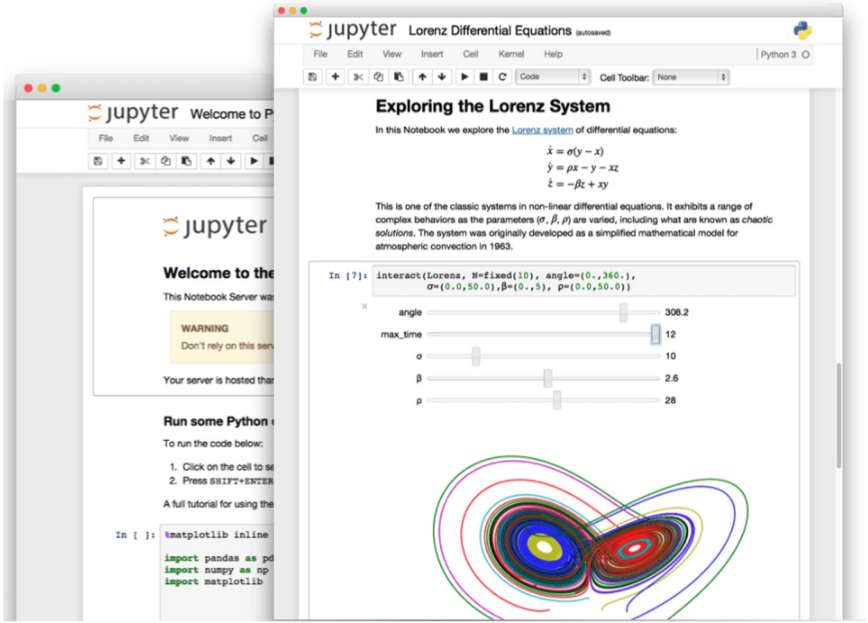
Jupyter Environment



- The Jupyter Notebook is an open-source web-based development application that allows you to create documents that can have code, equations, visualizations and narrative text.

- Mainly used for data cleaning and Data transformation, statistical modeling, data visualization, machine learning, deep learning and much more.

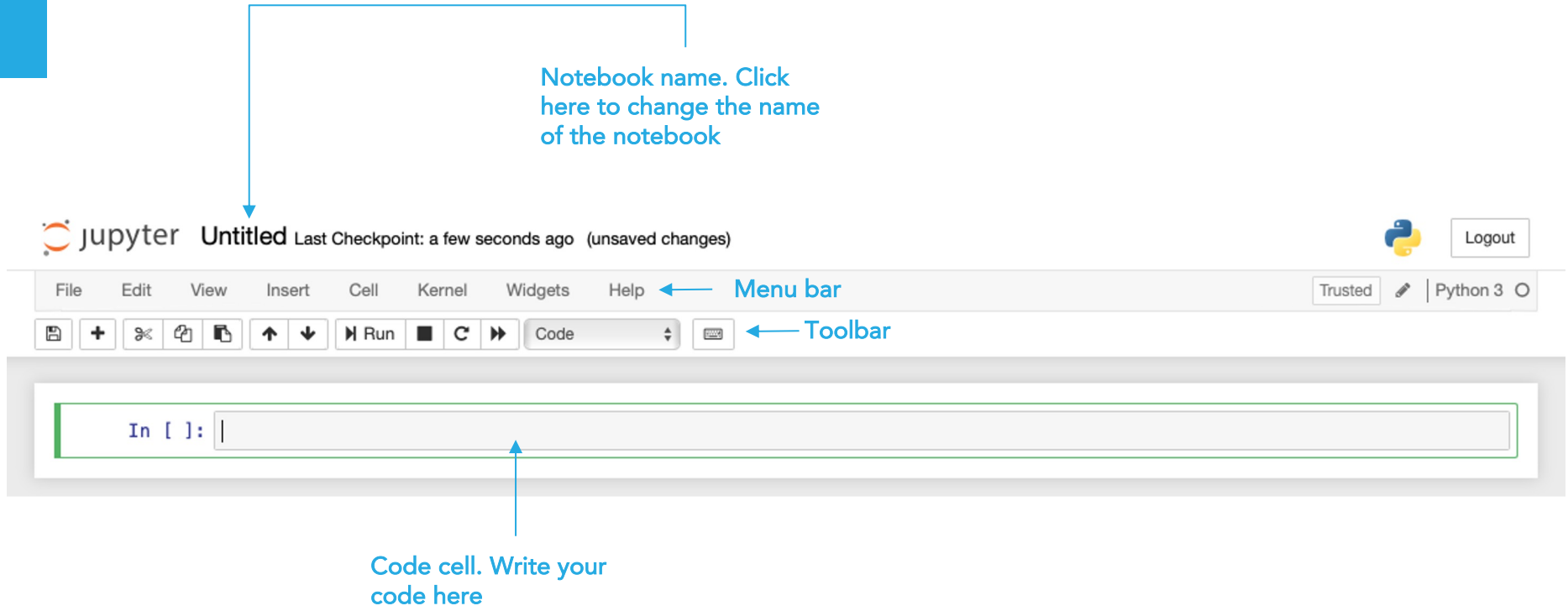
Jupyter Environment



- You can use Jupyter for over 40 programming languages. Within the data science community, mainly used for Python, R, Julia and Scala.

- You can also use Jupyter with big data tools, like Apache Spark from Python (known as PySpark), R and Scala. You can explore big data with pandas, scikit-learn, ggplot2, TensorFlow and more.

Jupyter Editor



Variables

- Variables are containers that store data. Python has no command for declaring variable
- You simply assign a value to a variable to create it
- You use = to assign a value to a variable

```
age = 21

name = "Mary"

print(age)
print(name)
```

```
21
Mary
```

Print variables with texts

```
print(name, "is", age, "years old")
```

```
Mary is 21 years old
```

```
print("John is older than", name, "He is", age+5)
```

```
John is older than Mary He is 26
```

Reassign variable

```
friendname = name

print(friendname)
```

```
Mary
```

Multiple variables assignment with same value

```
mary_age = john_age = anil_age = 22

print("Mary's Age:", mary_age)
print("John's Age", john_age)
print("Anil's Age", anil_age)
```

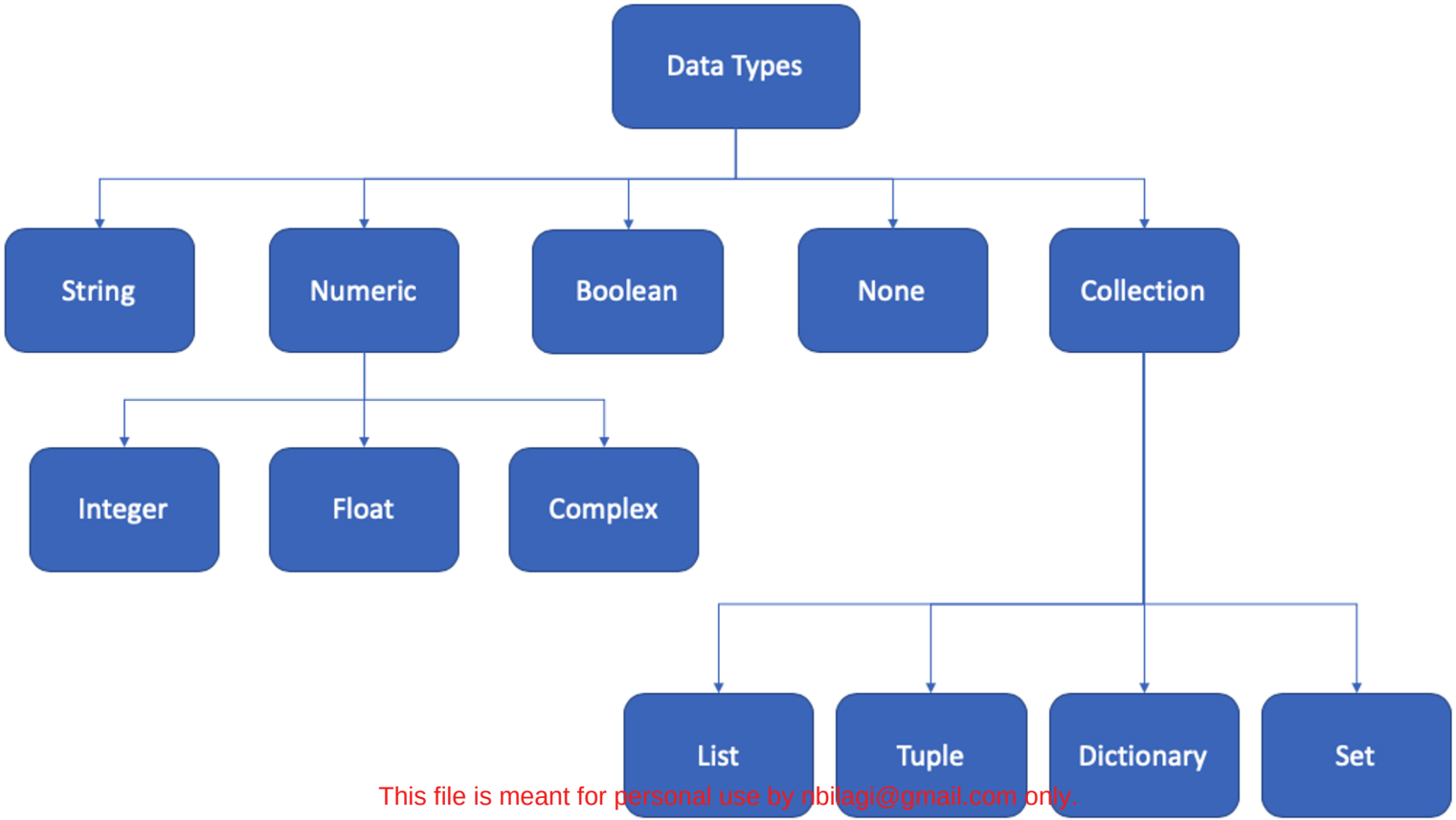
```
Mary's Age: 22
John's Age 22
Anil's Age 22
```

Multiple variables assignment with different values

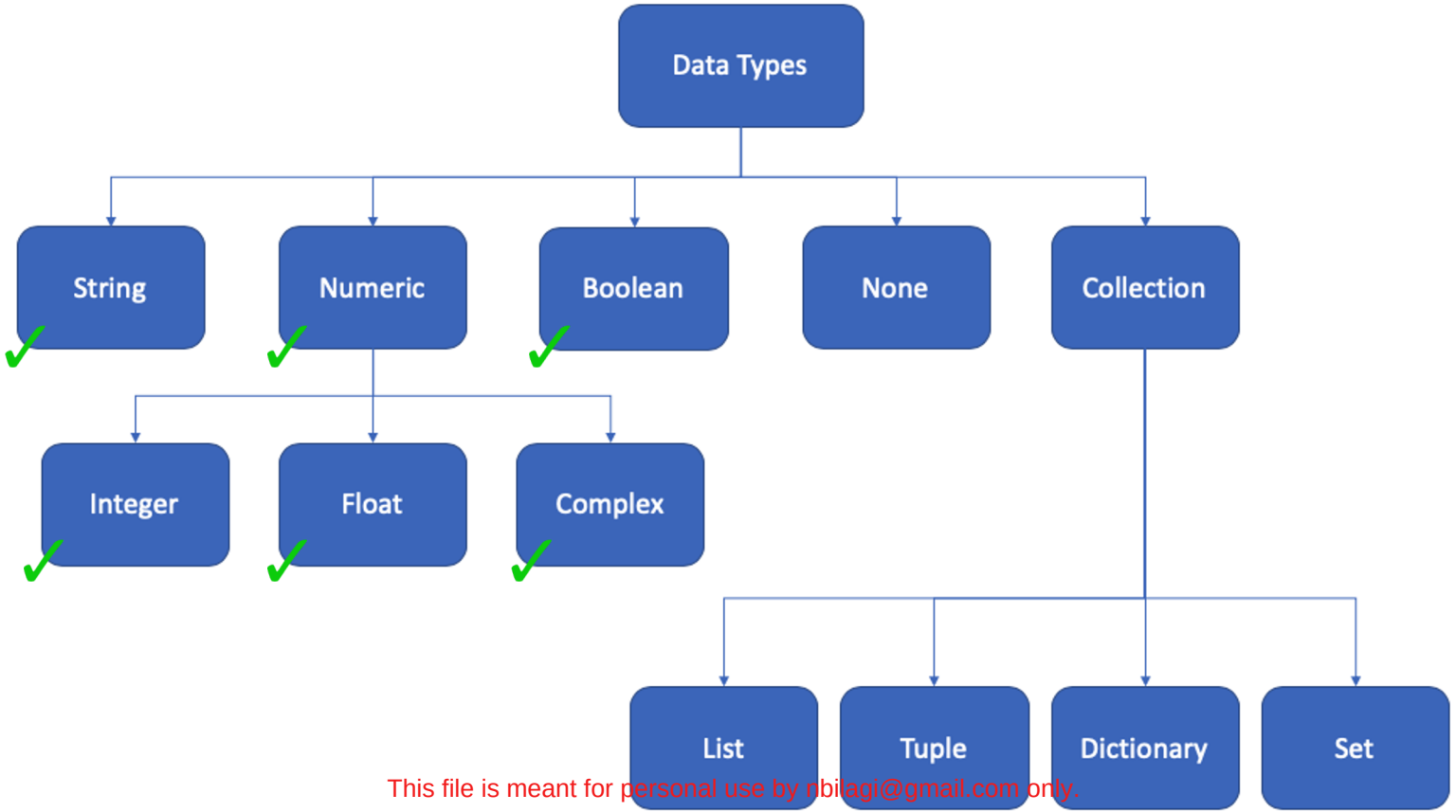
```
age, income, savings = 21, 4000, 200

print("Age:", age)
print("Income", income)
print("Savings", savings)
```

```
Age: 21
Income 4000
Savings 200
```

Python data types



We learn the following in our session today



Create numeric & string variables

```
age = 21
name = "Mary"
```

```
print(age)
print(name)
```

```
21
Mary
```

Create float, complex & boolean variables

```
# create a float variable
product_price = 3423.45

# create a boolean variable
success = True
```

```
print(product_price)
print(success)
```

```
3423.45
True
```

Checking data types with type() function

```
type(age)
```

```
int
```

```
type(name)
```

```
str
```

```
type(product_price)
```

```
float
```

```
type(success)
```

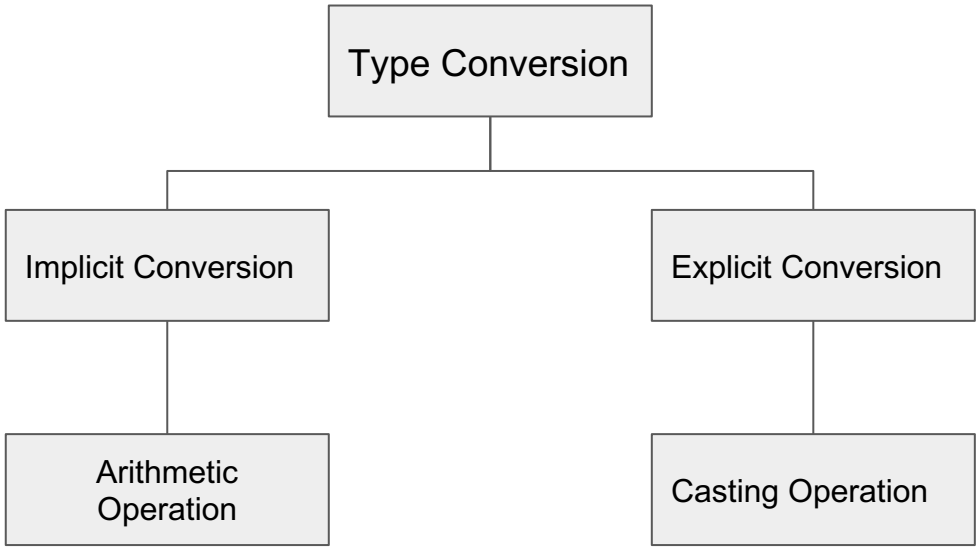
```
bool
```

Data type conversion

Python allows to convert one data type to another

Implicit Conversion: Conversion done by Python interpreter with programmer's intervention

Explicit Conversion: Conversion that is user-defined that forces an expression to be of specific data type



Implicit conversion

```
# Integer type data
income = 200

# Float type data
additional_income = 50.75

# Implicit conversion. Upon addition the resultant variable becomes float type
total_income = income + additional_income

print(total_income)
print(type(total_income))
```

```
250.75
<class 'float'>
```

Explicit conversion

```
price = 100

tax = "18"

total_cost = price + tax
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-52-6c276f72b675> in <module>
      3 tax = "18"
      4
----> 5 total_cost = price + tax

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

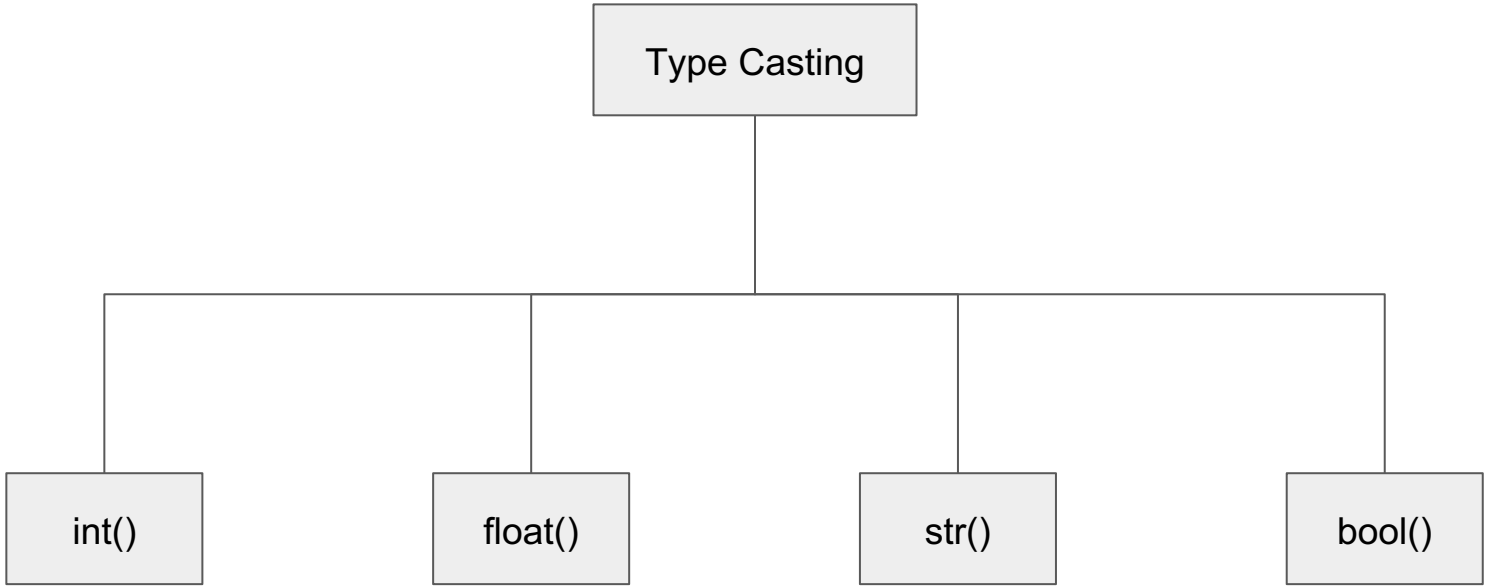
Explicit conversion

```
price = 100

tax = "18"

total_cost = price + int(tax)
```

```
print(total_cost)
print(type(total_cost))
```

```
118
<class 'int'>
```

Type casting



Type casting

```
income = 10000.45
print(int(income))

age = 25
print(float(age))
```

```
10000
25.0
```

```
price = 34.6
str(price)
```

```
'34.6'
```

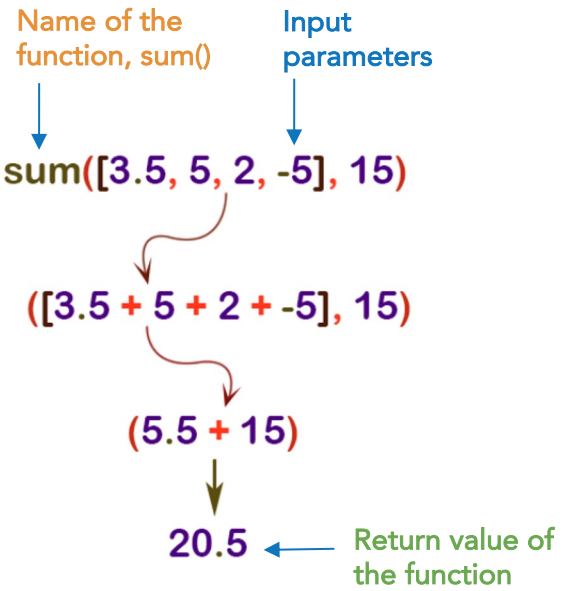
```
x, y = 0, 10
print(bool(x))
print(bool(y))
```

```
False
True
```

Functions in Python

What is a function?

- A function is a block of code that runs when called
- A function has a name. You call the function by its name
- A function can take input(s), known as input parameters
- A function can return data as a result



The print() Function

The print() in python

Great Learning

Words or sentences separated by a comma within a print() function get concatenated when printed.

```
print("Hello World")
```

```
Hello World
```

```
print("Hello", "how are you?")
```

```
Hello how are you?
```

```
fruits = ("avocado", "apple", "cherry")
print(fruits)
```

```
('avocado', 'apple', 'cherry')
```

The print() in python

The sep is an optional parameter. When output is printed, each word is separated by ^^^ characters

```
print("Hello", "how are you?", sep=" ^^^ ")
```

```
Hello ^^^ how are you?
```

```
print("Hello World")
print("\n")
print("How are you?")
```

```
Hello World


How are you?
```

print('\n') gives a new blank line

The backslash "\" is known as escape character. It is used in representing certain whitespaces.

For example '\t' is a tab and '\n' is a new line.

Great Learning

Invalid use of opening and closing quotes

```
print('Welcome, Lets learn python")
```

```
  File "<ipython-input-17-3d5d6be15599>", line 1
    print('Welcome, Lets learn python")
                                       ^
SyntaxError: EOL while scanning string literal
```

Print with concatenation

```
print("Welcome" + "To the world of python")
```

```
WelcomeTo the world of python
```

Great Learning

You cannot concatenate string & number

```
print("Your age is " + 35)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-0aae4f0f9b86> in <module>
----> 1 print("Your age is " + 35)

TypeError: can only concatenate str (not "int") to str
```

That's wrong because adding numbers to strings doesn't make any sense. You need to explicitly convert the number to string first, in order to join them together.

Concatenating with type casting

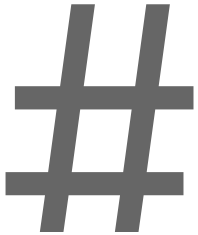
```
print("Your age is " + str(35))
```

```
Your age is 35
```


Explicit conversion of a number to string with str() function.

We learn more such type conversions in our upcoming sessions

Commenting code

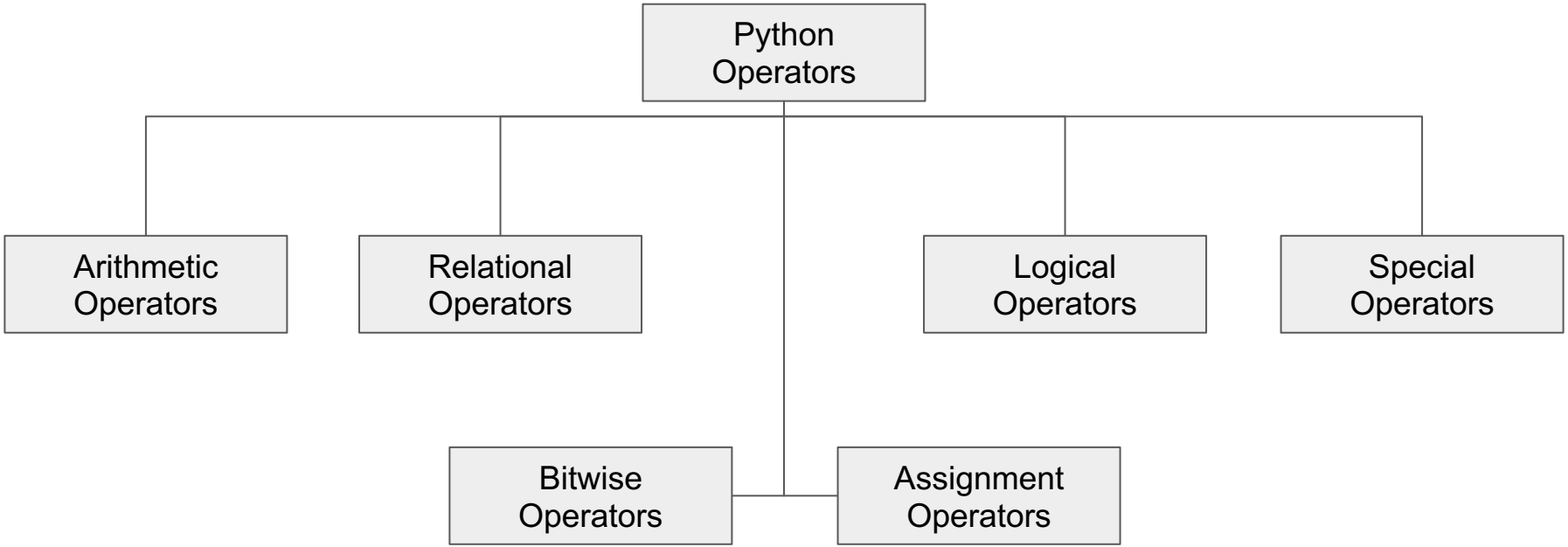


```
# This is a single line comment
print('Hello')

# This is a single line comment
n=10
print(n)


''' This is a multiline comment
which can be extended more than
one line'''
print('Hello')
```

Code commenting is required in order to describe the purpose of the code. This is not only a good practice but also a mandatory practice in most organizations.

Python Operators

Python operators



Arithmetic operators: Addition

```
# Most common Arithmetic Operators are:
        + (Addition)
        - (Subtraction)
        * (Multiplication)
        / (Division)
        % (Modulus)
        ** (Square)
```

```
# Adding 2 integer variables
price = 100
tax = 25
total_cost = price + tax
print(total_cost)
```

```
125
```

```
# Adding1 integer variable and 1 float variable
price = 100
tax = 12.80
total_cost = price + tax
print(total_cost)
```

```
112.8
```

Arithmetic operators: Addition

```
1  # Adding two strings
2  first_name = "Mike"
3  second_name = "Anderson"
4  full_name = first_name+ " " + second_name
5  print(full_name)
```

```
Mike Anderson
```

```
1  # Adding two strings
2  text = "Age is "
3  age = "22"
4  combine = text + age
5  print(combine)
```

```
Age is 22
```

Note that 22 is a string here because it has been put within the quotes.

Here two strings are getting concatenated.

Arithmetic operators: Subtraction

```
# subtracting two integer variables
price = 120
discount = 35
net_price = price - discount
print(net_price)
```

```
85
```

```
# subtracting one integer variable and one float variable
price = 125
discount = 35.5
net_price = price - discount
print(net_price)
```

```
89.5
```

Arithmetic operators: Subtraction

```
1 # Subtracting two strings
2 first_name = "Mike"
3 second_name = "Anderson"
4 full_name = first_name - second_name
5 print(full_name)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-997d527bdf02> in <module>()
      2 first_name = "Mike"
      3 second_name = "Anderson"
----> 4 full_name = first_name - second_name
      5 print(full_name)

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

When two strings are added, the operator basically concatenates the two strings. Subtracting two strings does not make any sense.

Arithmetic operators: Multiplication

```
# Multiplication of 2 integer variables
price = 120
quantity = 15
total_cost = price * quantity
print(total_cost)
```

```
1800
```

```
# Multiplication of 1 integer & 1 float variable
price = 120.50
quantity = 15
total_cost = price * quantity
print(total_cost)
```

```
1807.5
```

Arithmetic operators: Multiplication

```
# Multiplication of string with integer
value = "India"
count = 3
result = value * count
print(result)
```

```
IndiaIndiaIndia
```

Arithmetic operators: Multiplication

```
# Multiplication of 2 strings not possible
value = "Boat"
count = "3"
result = value * count
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-77-25575f6e63ae> in <module>
      2 value = "Boat"
      3 count = "3"
----> 4 result = value * count
      5 print(result)

TypeError: can't multiply sequence by non-int of type 'str'
```

Arithmetic operators: Division

```
# Division using 2 integer variables
total_cost = 2000
quantity = 100
price_per_unit = total_cost / quantity
print(price_per_unit)
```

```
20.0
```

```
# Division using 1 integer & 1 float variable
total_cost = 1560.75
quantity = 100
price_per_unit = total_cost / quantity
print(price_per_unit)
```

```
15.6075
```

Arithmetic operators: Division

```
# Cannot divide a string by a number
service = "airlines"
value = 3
result = service / value
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-7c4127a42622> in <module>
      2 service = "airlines"
      3 value = 3
----> 4 result = service / value
      5 print(result)

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

Arithmetic operators: Get quotient

```
# division operation to get quotient
# it is known as floor division
total_cost = 13000
quantity = 23
price_per_unit = total_cost // quantity
print(price_per_unit)
```

```
565
```

```
# division operation to get quotient
# it is known as floor division
total_cost = 13000.50
quantity = 23
price_per_unit = total_cost // quantity
print(price_per_unit)
```

```
565.0
```

Arithmetic operators: Get remainder

```
# Using modulus operator to find the remainder
total_cost = 2700
quantity = 23
price_per_unit = total_cost % quantity
print(price_per_unit)
```

```
9
```

```
# Using modulus operator to find the remainder
total_cost = 2700.50
quantity = 23
price_per_unit = total_cost % quantity
print(price_per_unit)
```

```
9.5
```

Arithmetic operators: Get square & power output

```
# squaring varibales
width_of_square = 20
area = width_of_square * width_of_square
print(area)
```

```
400
```

```
# power output
value = 2
n = 3

# value raised to n
result = value ** 3
print(result)
```

```
8
```

Runtime variable

```
# runtime variable
yourname = input("Enter your name:")
print("Welcome", yourname)
```

```
Enter your name:Steve
Welcome Steve
```

```
type(yourname)
```

```
str
```

Runtime variable

```
price = input("Enter price:")
quantity = input("Enter quantity")
total_cost = price * quantity
print(total_cost)
```

```
Enter price:34
Enter quantity34

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-130-c4a9ee6d0895> in <module>
      1 price = input("Enter price:")
      2 quantity = input("Enter quantity")
----> 3 total_cost = price * quantity
      4 print(total_cost)

TypeError: can't multiply sequence by non-int of type 'str'
```

Relational operators

Relational operators are used to compare 2 values and take certain decisions based on the outcome

<	is less than
<=	is less than & equal to
>	is greater than
>=	is greater than & equal to
==	is equal to
!=	is not equal to

Relational operators

```
mona_age = 35
john_age = 25
```

```
mona_age > john_age
```

```
True
```

```
mona_age < john_age
```

```
False
```

```
mona_age == john_age
```

```
False
```

```
mona_age >= john_age
```

```
True
```

```
mona_age <= john_age
```

```
False
```

```
mona_age != john_age
```

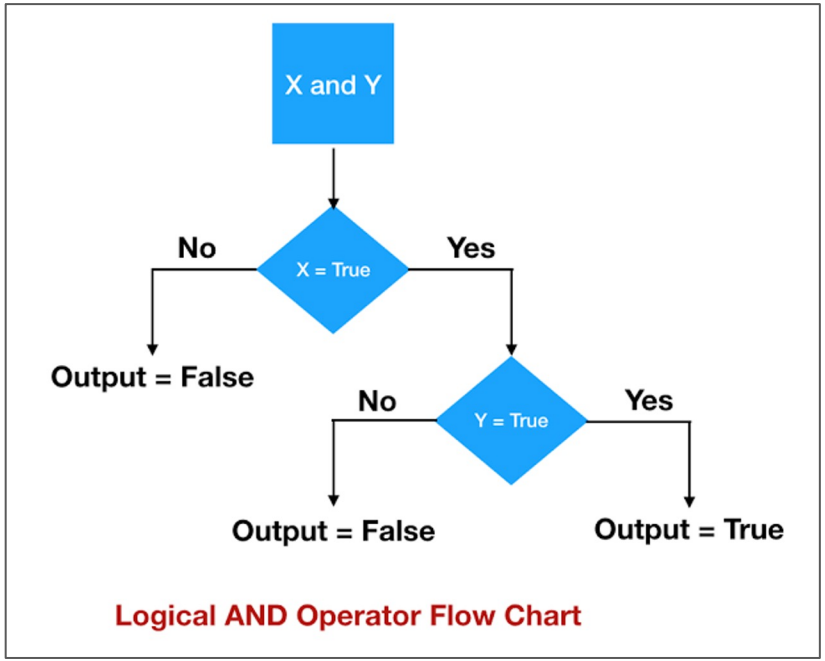
```
True
```

Logical operators

Logical operators in Python are used for conditional statements are either True or False

AND	Returns True if both the operands are True
OR	Returns True if either of the operands are True
NOT	Returns True if the operand is False

AND operator

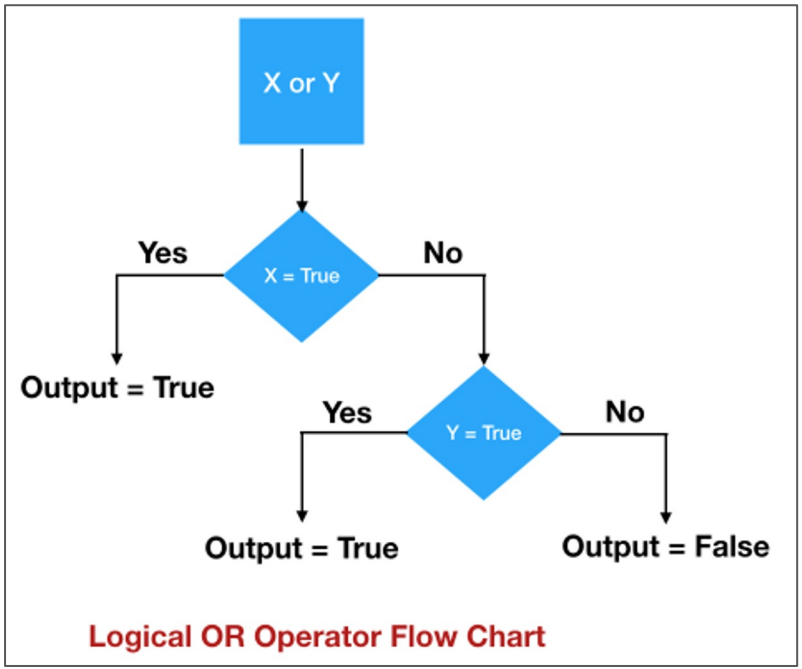



```
x = True
y = False
```

```
x and y
```

```
False
```

OR operator



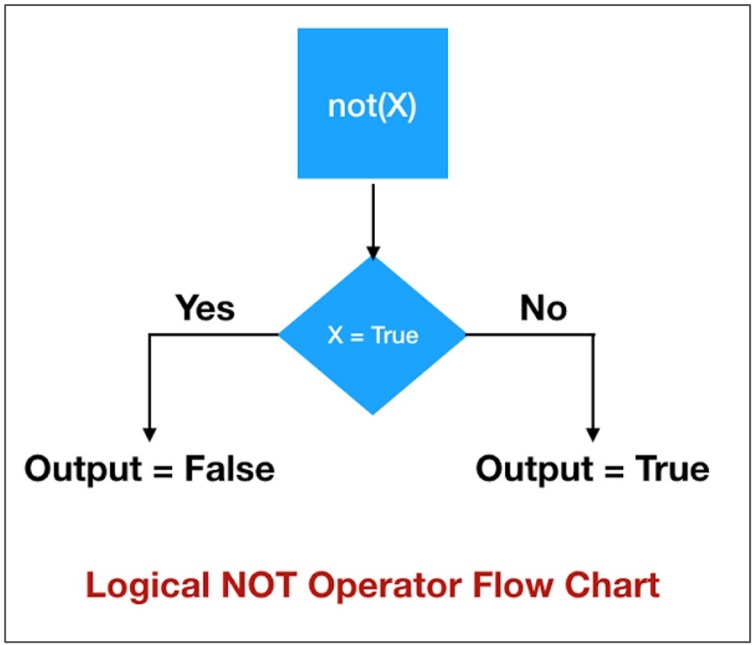
Logical OR Operator Flow Chart

```
x = True
y = False
```

```
x or y
```

```
True
```

NOT operator




```
x = True
y = False
```

```
not x
```

```
False
```

Membership operators

Membership operators tests whether a value is a member of a sequence. The sequence may be a list, a string, a tuple, or a dictionary.

in

The ‘in’ operator is used to check if a value exists in any sequence object or not. For example, if 2 exists in a list, say [4, 5, 7, 2]. This evaluates to True if it finds a value in the specified sequence object. Otherwise it returns a False.

Membership operators

not in

A ‘not in’ works in an opposite way to an ‘in’ operator. A ‘not in’ evaluates to True if a value is not found in the specified sequence object. Else it returns a False.

Membership operators

Example:

in operator:

```
string = "Hello World"
var = 'o'
print(var in string)
```

```
True
```

```
string = "Hello World"
var = 'g'
print(var in string)
```

```
False
```

not in operator:

```
string = "Hello World"
var = 'o'
print(var not in string)
```

```
False
```

```
string = "Hello World"
var = 'g'
print(var not in string)
```

```
True
```

Some bitwise operators

Python Bitwise Operators take one to two operands, and operates on it/them bit by bit, instead of whole

& (Bitwise and)	The binary and (&) takes two values and performs an AND-ing on each pair of bits.
| (Bitwise or)	Compared to &, this one returns 1 even if one of the two corresponding bits from the two operands is 1.

The truth tables

&
(Bitwise and)

Value	Value	Value & Value
True	True	True
True	False	False
False	True	False
False	False	False

|
(Bitwise or)

Value	Value	Value | Value
True	True	True
True	False	True
False	True	True
False	False	False

Some bitwise operators

Great Learning

Example:

& operator:

```
# & operation examples:
number = 9

(number % 3 == 0) & (number % 5 == 0)
```

```
False
```

(9 % 3 == 0) & (9 % 5 == 0)

Implies True & False

This results in Fasle.

| operator:

```
# | operation examples:
number = 9

(number % 3 == 0) | (number % 5 == 0)
```

```
True
```

(9 % 3 == 0) | (9 % 5 == 0)

Implies True | False

This results in True.

More assignment operators

The assignment operators are used to store data into a variable.

```
a += b  is same as a =  a + b
a *= b  is same as a =  a * b
a /= b  is same as a =  a / b
a %= b  is same as a =  a % b
a **= b is same as a =  a ** b
a //= b is same as a =  a // b
```

Data slicing

```
a = 'great learning'
```

```
# get only the first element
# indexing starts from 0 in python
a[0]
```

```
'g'
```

```
# display all the letters in the variable
a[:]
```

```
'great learning'
```

```
# display all the letter using len()
# len(): gives length of the variable
a[:len(a)]
```

```
'great learning'
```

':' is used to specify range of the sequence to be sliced

Data slicing

```
# display the first three letters alone by specifying the start and the end
# indexing starts from 0 in python, ending at n-1
a[0:3]
```

```
'gre'
```

```
# display from the third letter till the last
a[2:]
```

```
'eat learning'
```

```
# display from the 3rd letter till 10th letter
a[2:11]
```

```
'eat learn'
```

Data slicing

```
# displaying the list from the second letter till the second last letter
a[1:-1]
```

```
'reat learnin'
```

```
# reverse the string
a[::-1]
```

```
'gninrael taerg'
```

```
# reverse the string and displace the third last letter.
a[::-1][-3]
```

```
'e'
```

String Handling

```
# replace 'g' with 'G' in the string
a.replace('g','G')
```

```
'Great learninG'
```

```
# convert the string in upper case
a.upper()
```

```
'GREAT LEARNING'
```

```
# convert the string in lower case
a.lower()
```

```
'great learning'
```

```
# convert the first letter of each string to upper case
a.title()
```

```
'Great Learning'
```

String Handling

```
1 # Concatenate each element of s1 with sep
2 s1 = "08", "03","2000"
3 sep = '-'
4 sep.join(s1)
```

```
'08-03-2000'
```

Returns a concatenated string where each character of s1 is separated with sep string which is '-'

String Handling

```
1 # Split the string, text, using comma as a separator:
2 text = "Cereals Grocery Cosmetics"
3 text.split(' ')
```

```
['Cereals', 'Grocery', 'Cosmetics']
```

Returns a list containing elements as strings separated by the space character

String Handling

```
1 # Print the string by removing leading and trailing "*" character
2 text = "**Hello**World**"
3 text.strip("*")
```

```
'Hello**World'
```

```
1 # Print the string by removing leading "*" character
2 text = "**Hello World"
3 text.strip("*")
```

```
'Hello World'
```

```
1 # Print the string by removing trailing "*" character
2 text = "Hello World**"
3 text.strip("*")
```

```
'Hello World'
```

Returns a copy of the string with both leading and trailing characters removed (based on the string argument passed)

Python Flow Control

Python Flow Control

Any program has a flow. The flow is the order in which the program's code executes. The control flow of a Python program is controlled by:

1. Conditional Statements
2. Loops
3. Function Calls

We cover the basics of conditional statements and loops in today's session

The if-statement

Sometimes we want to execute a code only if a certain condition is true.

The *if* statement is used in Python for decision making. An "if statement" is written by using the *if* keyword.

Syntax:

```
if test expression:
    statement(s)
```

The if-statement

```
num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
```

```
3 is a positive number.
This is always printed.
```

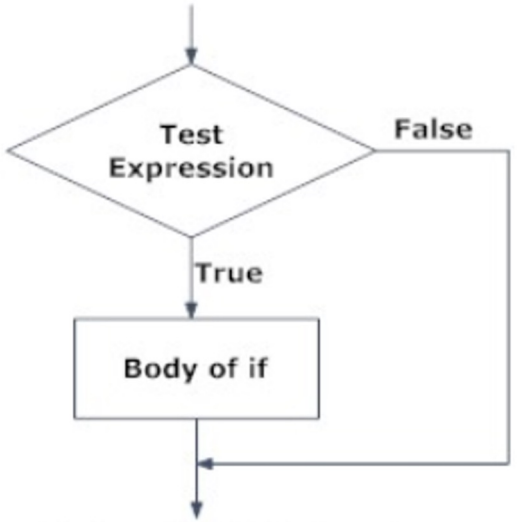


Fig: Operation of if statement

The if-else statement

The *'if..else'* statement evaluates a *test expression* and will execute the code that is part of the *'if'* expression if the *test expression True*.

If the test expression is *False*, the code that is part of the *'else'* expression is executed. Note the indentation that is used to separate the 'if' and 'else' blocks.

Syntax:

```
if test expression:
    Body of if
else:
    Body of else
```

The if-else statement

```
num = -1
if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

```
Negative number
```

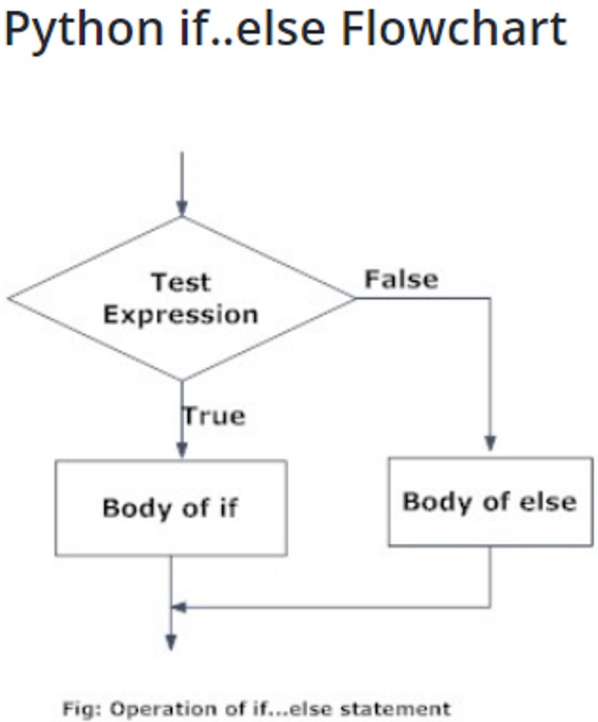


Fig: Operation of if...else statement

The *while* Loop

Great Learning

Loops are used to repeat the execution of a specific block of code.

The '*while loop*' in Python is used to iterate over a block of code as long as the test expression holds true.

We generally use this loop when the number of times to iterate is not known to us beforehand.

Syntax:

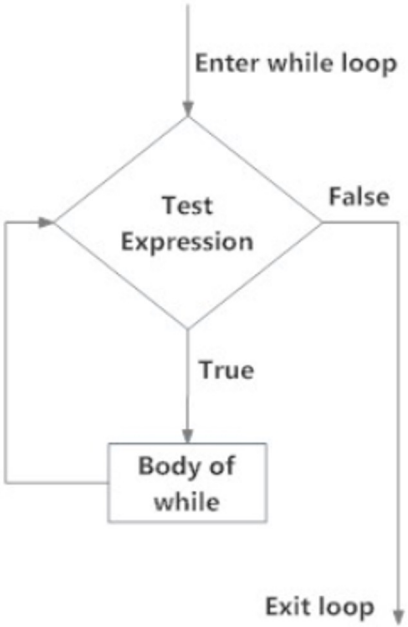
```
while test_expression:
    Body of while
```

The *while* Loop

```
n = 5

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

```
The sum is 15
```

Flowchart of while Loop



The *for* Loop

The '*for loop*' in Python is used to iterate over the items of a sequence object like list, tuple, string and other iterable objects.

The iteration continues until we reach the last item in the sequence object. Note the indentation that is used in a '*for loop*' to separate the rest of the code from the 'for loop' syntax

Syntax:

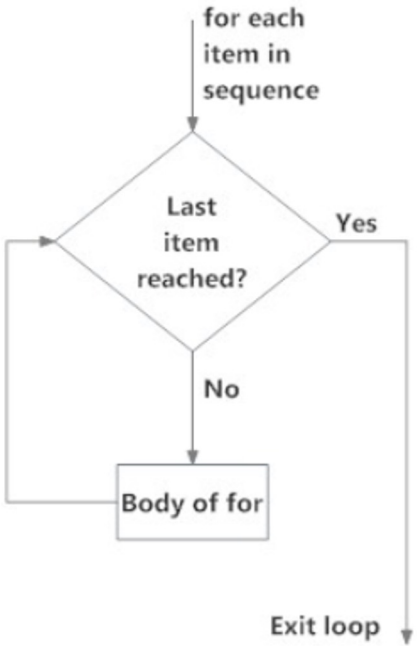
```
for i in sequence:
    Body of for
```

The *for* Loop

```
numbers = range(0,10)
sum = 0
for i in numbers:
    sum = sum+i

# Output: The sum is 48
print("The sum is", sum)
```

```
The sum is 45
```

Flowchart of for Loop



Pseudocode

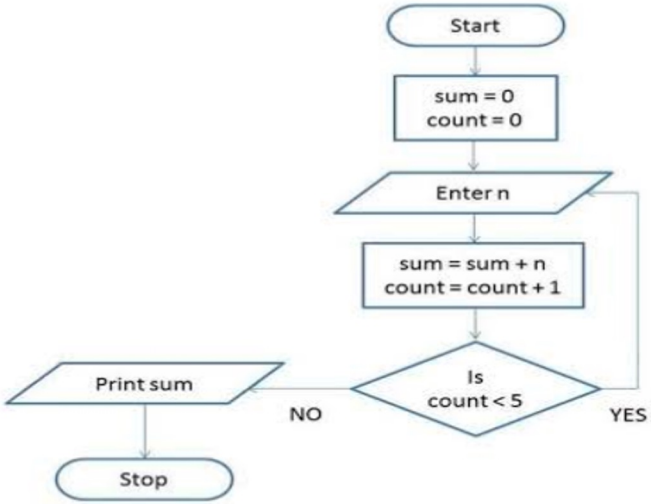
What is Pseudocode?

Pseudocode is a step-by-step written outline of your code that you can gradually transcribe into programming language

Example: Pseudocode

Find the sum of 5 numbers:

1. Set sum=0, count=0
2. Enter the number
3. Add number to sum and store it back to sum
4. Increment count by 1
5. If count < 5, go to step 2 else print total



do it YOURSELF

Great Learning

Write pseudocode to calculate sum & average of 10 numbers that you input

did you know?

There exists a word “Pythonic”? What does it mean?

There are many ways to accomplish the same task in Python, but there is usually one preferred way to do it. This preferred way is called "pythonic."

Read The Hitchhiker’s Guide to Python (https://guide.org/writing/style/).

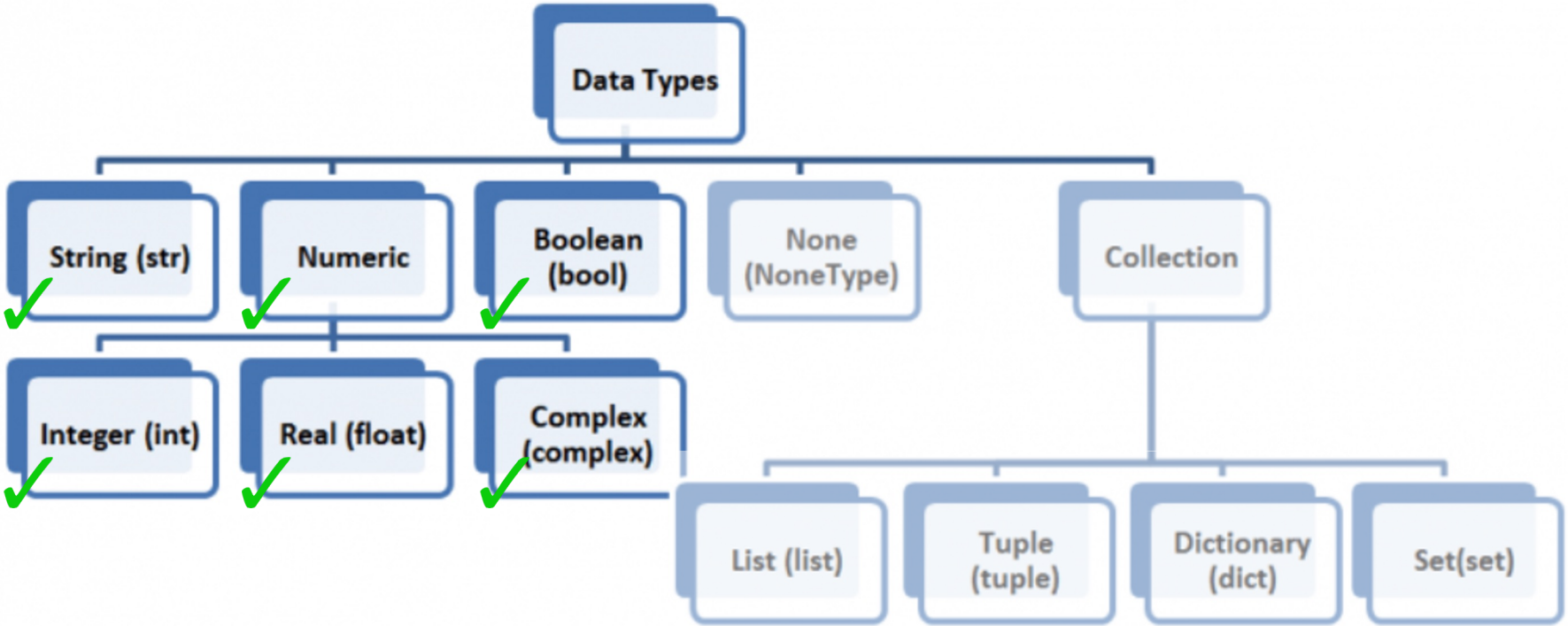
Python Objects

Container or Collection Objects

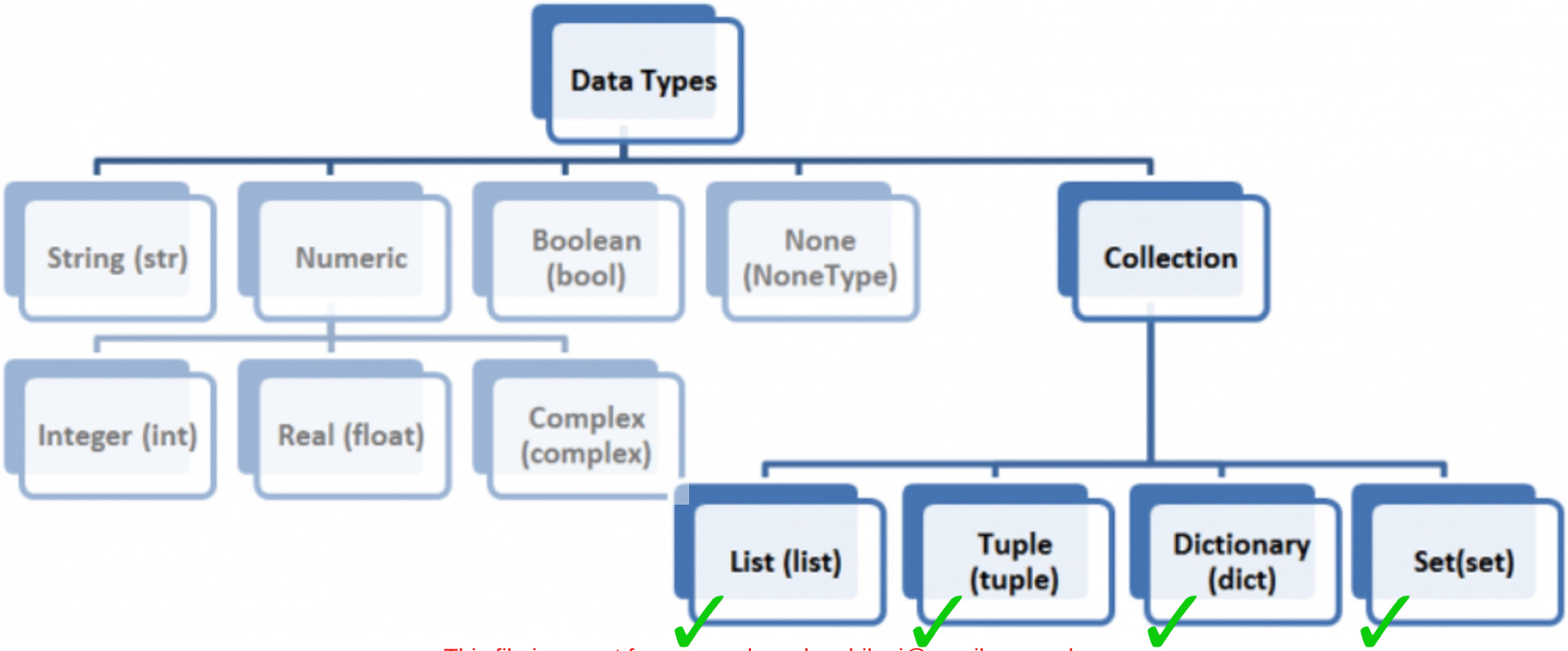
Container or collection objects

- Container or collection objects are objects that can hold any number of arbitrary objects
- Container provides a way to access the child objects and iterate over them
- Container or collection objects:
  - Dictionary
  - Tuple
  - List
  - Set

In our last session we covered...



We learn the following in our session today



List

List

- A list is a container object
- Can have duplicates unlike a Set
- Homogeneous
- Supports the following:
  - Append new elements
  - Concatenate
  - Access using indexes
  - min(), max()
  - In, not in

Creating lists

```
# Creating Lists
numbers = range(1,20)
print(numbers[0])
print(numbers[2])

characters = ["Python", "Scala", "Spark"]
print(characters[0])
print(characters[1])
```

```
1
3
Python
Scala
```

```
# Concatenating Lists
states_in_india = ["Tamil Nadu", "Karnataka", "Haryana"]
states_in_us = ["California", "Florida", "Texas", "Alabama"]
print(states_in_india + states_in_us)
```

```
['Tamil Nadu', 'Karnataka', 'Haryana', 'California', 'Florida', 'Texas', 'Alabama']
```

```
# Append
states_in_india.append("Kerala")
print(states_in_india)
```

```
['Tamil Nadu', 'Karnataka', 'Haryana', 'Kerala']
```

Sorting a list using the built-in sort() function

```
# Sorting a list
a = [10, 12, 11, 16, 71, 12, 9, 56]
b = [11, 16, 2, 34, 21, 11, 8, 18]
c = ["Zebra", "Apple", "Anchor", "Assets", "Baseball", "Basket"]
```

```
# Using built-in sort function
print(a)
a.sort()
print(a)
```

```
[10, 12, 11, 16, 71, 12, 9, 56]
[9, 10, 11, 12, 12, 16, 56, 71]
```

```
# Sorting text in a list
print(c)
c.sort()
print(c)
```

```
['Zebra', 'Apple', 'Anchor', 'Assets', 'Baseball', 'Basket']
['Anchor', 'Apple', 'Assets', 'Baseball', 'Basket', 'Zebra']
```

Sort the list in reverse order by passing the reverse=True parameter to the sort() function

```
# Sorting a list
a = [10, 12, 11, 16, 71, 12, 9, 56]
b = [11, 16, 2, 34, 21, 11, 8, 18]
c = ["Zebra", "Apple", "Anchor", "Assets", "Baseball", "Basket"]
```

```
# Using built-in sort function
print(a)
a.sort()
print(a)
```

```
[10, 12, 11, 16, 71, 12, 9, 56]
[9, 10, 11, 12, 12, 16, 56, 71]
```

Sorting a list using the sorted() function

```
# using sorted() function
print(b)
bs = sorted(b)
print(bs)
```

```
[11, 16, 2, 34, 21, 11, 8, 18]
[2, 8, 11, 11, 16, 18, 21, 34]
```

do it YOURSELF

Great Learning

Sort the list in reverse order by passing the reverse=True parameter to the sorted() function

Hint: sorted(b, reverse=True)

```
# using sorted() function
print(b)
bs = sorted(b)
print(bs)
```

```
[11, 16, 2, 34, 21, 11, 8, 18]
[2, 8, 11, 11, 16, 18, 21, 34]
```

More list operations

```
# given 2 lists which have only numbers
List1 = [1,2,3,4]
List2 = [5,6,7,8]
Concatenation = List1 + List2
Concatenation
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
# given 2 list which have numbers and string
List1 = [1,2,3,4]
List2 = ["a","b"]
Concatenation = List1 + List2
Concatenation
```

```
[1, 2, 3, 4, 'a', 'b']
```

```
# list with mixed data type and add 2 list
List1 = ["a",2,"b",4]
List2 = ["a","b",1,2]
Concatenation = List1 + List2
Concatenation
```

```
['a', 2, 'b', 4, 'a', 'b', 1, 2]
```

More list operations

```
# Check existence of element in a list
List = [1,2,3,4,5]
print(2 in List)
```

```
True
```

```
# Find the number of elements in a list
List = [1,2,3,4,5]
len(List)
```

```
5
```

```
# Iterating through a list
List = [1,2,3,4,5]
for i in List:
    print(List)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

More list operations

```
# Max value in a list
List = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
max(List)
```

```
7654
```

```
# Min value in a list
List = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
min(List)
```

```
1
```

```
# Type cast a string to a list
string = 'qwerty'
list(string)
```

```
['q', 'w', 'e', 'r', 't', 'y']
```

More list operations

```
# Append element to a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.append('pig')
List
```

```
# Clear a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.clear()
List
```

```
# Copy a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
new_list = List.copy()
List.append('pig')
print('Old List: ', List)
print('New List: ', new_list)
```

```
Old List:  ['Penguin', 'cat', 'Hippopotam', 'dog', 'rabbit', 'pig']
New List:  ['Penguin', 'cat', 'Hippopotam', 'dog', 'rabbit']
```

More list operations

```
# Count number of specific elements
List = ['a', 'e', 'i', 'o', 'i', 'u']
List.count('i')
```

```
2
```

```
# Extend a list
List1 = [1,2,3,4]
List2 = [5,6,7,8]
List1.extend(List2)
print('Language List: ', List1)
```

```
Language List:  [1, 2, 3, 4, 5, 6, 7, 8]
```

```
# Find the index position of an element
List = ['a', 'e', 'i', 'o', 'i', 'u']
List.index('i')
```

```
2
```

```
# Insert element at specific index position
List = ['a', 'e', 'i', 'u']
List.insert(3, 'o')

print('Updated List: ', List)
```

```
Updated List:  ['a', 'e', 'i', 'o', 'u']
```

More list operations

```
# Remove an element from a list by index position
List = ['a', 'e', 'i', 'u']
return_value = List.pop(3)
print('Return Value: ', return_value)
print('Updated List: ', List)
```

```
Return Value:  u
Updated List:  ['a', 'e', 'i']
```

```
# Remove an element from a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.remove('rabbit')
print('Updated list: ', List)
```

```
Updated list:  ['Penguin', 'cat', 'Hippopotam', 'dog']
```

```
# Reverse a list
List = ['Penguin','cat','Hippopotam', 'dog', 'rabbit']
List.reverse()
List
```

```
['rabbit', 'dog', 'Hippopotam', 'cat', 'Penguin']
```

We learnt the list object...

Create using [ ]

Editable

Tuple

Tuples

- Type of container object
- Known as sequence types
- Can have heterogeneous sequence of elements
- Immutable
- Elements are separated by comma, enclosed in ( ) parenthesis
- Supported operations:
  - In, Not in
  - Min(), Max()
  - Compare
  - Concatenate, Slice, Index

Creating tuples

```
# Create a tuple
my_tuple = (100, 250, "Robert")

# Access the elements in a tuple
print(my_tuple[0])
print(my_tuple[-1])
```

```
100
Robert
```

```
# YOU CANNOT CHANGE AN ELEMENT
my_tuple[0]=450
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-3a8a9921f822> in <module>()
      1 # YOU CANNOT CHANGE AN ELEMENT
----> 2 my_tuple[0]=450

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

Concatenate & slice tuple

```
# Concatenate two tuples
my_tuple = (100, 250, 650)
your_tuple = ("Daniel", 450, 200, 750, "Siva")

print(my_tuple+your_tuple)
```

```
(100, 250, 650, 'Daniel', 450, 200, 750, 'Siva')
```

```
#Slice a tuple, min(), max()
print(your_tuple[0])
print(your_tuple[2:3])
print(your_tuple[1:5])
print(min(my_tuple) + max(my_tuple))
```

```
Daniel
(200,)
(450, 200, 750, 'Siva')
750
```

Tuple operations

```
# Tuple Repetition
Tuple = (1,2,3,4)
Repetition = Tuple *2
Repetition
```

```
(1, 2, 3, 4, 1, 2, 3, 4)
```

```
# Concatenate a tuple
Tuple1 = (1,2,3,4)
Tuple2 = (5,6,7,8)
Concatenation = Tuple1 + Tuple2
Concatenation
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
```

```
# Checking existence of an element in a tuple
Tuple = (1,2,3,4,5)
print(2 in Tuple)
```

```
True
```

Tuple operations

```
# Length of a tuple
Tuple = (1,2,3,4,5)
len(Tuple)
```

```
5
```

```
# Iterating a tuple
Tuple = (1,2,3,4,5)
for i in Tuple:
    print(i)
```

```
1
2
3
4
5
```

Tuple operations

```
# Finding maximum
Tuple = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
max(Tuple)
```

```
7654
```

```
# Finding minimum
Tuple = [1,2,3,4,4,5,67,8,565,4,65,76,7654,654,6543,7654]
min(Tuple)
```

```
1
```

Tuple operations

```
# type casting - convert a string into a tuple
Tuple = 'qwerty'
tuple(Tuple)
```

```
('q', 'w', 'e', 'r', 't', 'y')
```

```
## Sorted - output will be sorted but the original data will not be changed
age = (40,8,35,5)
print(sorted(age))

# default is ascending order
print(sorted(age , reverse = True))
print(age)
```

```
[5, 8, 35, 40]
[40, 35, 8, 5]
(40, 8, 35, 5)
```

Tuple operations

```
## Sorted - output will be sorted but the original data will not be changed
age = (40,8,35,5)
print(sorted(age))

# default is ascending order
print(sorted(age , reverse = True))
print(age)
```

```
[5, 8, 35, 40]
[40, 35, 8, 5]
(40, 8, 35, 5)
```

```
# Slicing a tuple
age = (40,8,35,5)
print(age[1])
print(age[0:])
print(age[:len(age)])
print(age[:-1])
print(age[-0:-1])
```

```
8
(40, 8, 35, 5)
(40, 8, 35, 5)
(40, 8, 35)
(40, 8, 35)
```

Tuple operations

```
# Reversing a tuple
fruits = ("orange","apple","cherry","apple","grapes")

print("reverse string :",fruits[::-1])
print("list from 2nd element til last :",fruits[2:])
print("print last 3 element :",fruits[-3:])
```

```
reverse string : ('grapes', 'apple', 'cherry', 'apple', 'orange')
list from 2nd element til last : ('cherry', 'apple', 'grapes')
print last 3 element : ('cherry', 'apple', 'grapes')
```

```
# Add tuple
age = (40,8,35,5)
name = ('jack','jerry')

ad_tup = age + name
ad_tup
```

```
(40, 8, 35, 5, 'jack', 'jerry')
```

Tuple operations

```
## sub tuple tuples together, sub will not work
ad_tup = age - name
ad_tup
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-582e9fba70da> in <module>
      1 ## sub tuple tuples together, sub will not work
----> 2 ad_tup = age - name
      3 ad_tup

TypeError: unsupported operand type(s) for -: 'tuple' and 'tuple'
```

```
del age[1] # tuple cannot delete an object in it
del age # tuple can be deleted completly
age # its not available after deleting
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-05932a2264c9> in <module>
----> 1 del age[1] # tuple cannot delete an object in it
      2 del age # tuple can be deleted completly
      3 age # its not available after deleting

TypeError: 'tuple' object doesn't support item deletion
```

did you know?

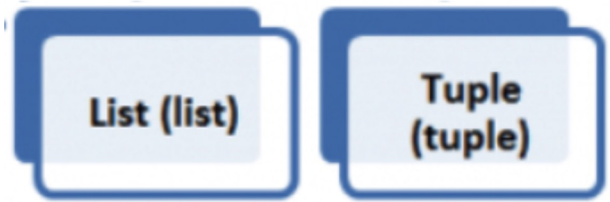
Due to mutability, you need more memory for lists and less memory for tuples.

WANT TO KNOW MORE?

To reduce memory fragmentation and speed up allocations, Python reuses old tuples. If a tuple no longer needed and has less than 20 items, instead of deleting it permanently Python moves it to a free list.

A free list is divided into 20 groups, where each group represents a list of tuples of length n between 0 and 20. Each group can store up to 2 000 tuples. The first (zero) group contains only 1 element and represents an empty tuple

We learnt list & tuple...



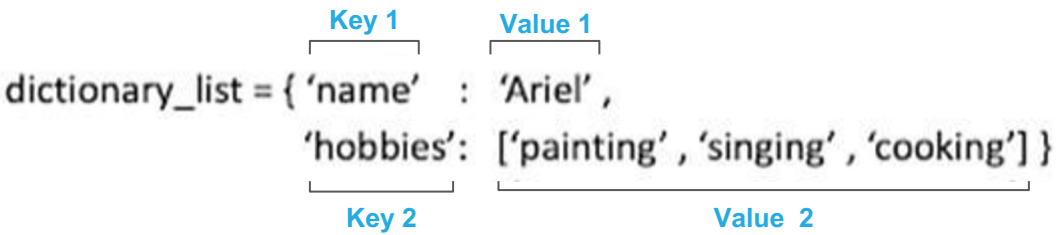
Create using	[ ]	( )
Editable	✎	⊘

Dictionary

Dictionary object

- A Python dictionary is a mapping of unique keys to values
- Use {} curly brackets to construct the dictionary
- [] square brackets to index it
- Dictionaries are mutable
- We will learn how to:
  - Access values in a dictionary
  - Update dictionary
  - Delete dictionary elements

A sample dictionary object



Dictionary object

```
# Create a NEW Dictionary object
books = {
    "R": 480,
    "Python": 650,
    "PySpark": 450,
    "Scala": 780,
    "Basic Stats": 650
}
```

```
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Scala': 780, 'Basic Stats': 650}
```

```
# Add elements to the books dictionary
books['Hadoop']=850
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Scala': 780, 'Basic Stats': 650, 'Hadoop': 850}
```

```
# Remove an element from the dictionary
del books['Scala']
print(books)
```

```
{'R': 480, 'Python': 650, 'PySpark': 450, 'Basic Stats': 650, 'Hadoop': 850}
```

```
# Check the length of the dictionary object
len(books)
```

```
5
```

Dictionary object

```
# Using Dictionary Objects

# Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()
```

```
['A', 'Quick', 'Brown', 'Fox', 'Jumps', 'over', 'the', 'Lazy', 'Dog']
```

```
# Initializa a dictionary object
# Use {} curly brackets to construct a dictionary object
my_dictionary = {}
```

```
# We will perform a word count using the dictionary object
for word in my_text.split() :
    if word not in my_dictionary :
        my_dictionary[word]=1
    else:
        my_dictionary[word]+=1

# The above code builds a frequency table for every word
```

```
# Print the output
# Output ia  key-value pair
print(my_dictionary)
```

```
{'A': 1, 'Quick': 1, 'Brown': 1, 'Fox': 1, 'Jumps': 1, 'over': 1, 'the': 1, 'Lazy': 1, 'Dog': 1}
```

Dictionary object: Default dictionary

```
# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = {}

# We will perform a word count using the dictionary object
for word in my_text.split() :
#    if word not in my_dictionary :
#        my_dictionary[word]=1
#    else:
       my_dictionary[word]+=1
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-47206380e9a7> in <module>()
     12 #        my_dictionary[word]=1
     13 #    else:
---> 14           my_dictionary[word]+=1
     15

KeyError: 'A'
```

Dictionary object: Default dictionary

- Import the defaultdict from collections module
- Initialize the dictionary object with defaultdict()
- Note: We passed int() to the defaultdict()
- When the dictionary object encounters a key that was not seen before, it initializes the key with a value returned by int(), in this case 0 (zero)

Dictionary object: Default dictionary

- Import the defaultdict from collections module
- Initialize the dictionary object with defaultdict()
- Note: We passed int() to the defaultdict()
- When the dictionary object encounters a key that was not seen before, it initializes the key with a value returned by int(), in this case 0

```
# Import for the defaultdict from collections module
from collections import defaultdict

# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = defaultdict(int)

# We will perform a word count using the dictionary object
for word in my_text.split() :
#    if word not in my_dictionary :
#        my_dictionary[word]=1
#    else:
       my_dictionary[word]+=1

print(my_dictionary)
```

```
defaultdict(<class 'int'>, {'A': 1, 'Quick': 1, 'Brown': 1, 'Fox': 1, 'Jumps': 1, 'over': 1, 'the': 1, 'Lazy': 1, 'Dog': 1})
```

Loop through the dictionary object

- Use keys() to loop through the keys in the dictionary object
- Use values() to loop through the values in the dictionary object

```
# Import for the defaultdict from collections module
from collections import defaultdict

# # Define a variable
my_text = "A Quick Brown Fox Jumps over the Lazy Dog"

# See how split() works
my_text.split()

my_dictionary = defaultdict(int)

# We will perform a word count using the dictionary object
for word in my_text.split() :
    my_dictionary[word]+=1

# Using key(), value() functions of dictionary obejct
for key, value in my_dictionary.items():
    print(key, value)
```

Built-in dictionary functions

```
# Checking length of dictionary object
dict = {'Name': 'vema', 'Age': 27};
len(dict)
```

```
2
```

```
# Converts the dictionary into the printable string representation
dict = {'Name': 'vema', 'Age': 27};
str(dict)
```

```
"{'Name': 'vema', 'Age': 27}"
```

```
# Checking the type of object
dict = {'Name': 'vema', 'Age': 27};
type(dict)
```

```
dict
```

Built-in dictionary functions

```
employee = {"Name": "Johny", "Age": 32}
employee.clear()
employee
```

```
{}
```

```
employee = {"Name": "Johny", "Age": 32}
new_employee = employee.copy()
new_employee
```

```
{'Name': 'Johny', 'Age': 32}
```

```
# seq is a tuple
seq = ('name', 'age', 'sex')
# tuple is type casted into dict and all the elements are converted into keys. So the key will not have any values.
dict = dict.fromkeys(seq)

print ("New Dictionary : ",  str(dict))

# assign a default value as 10 for all the keys
dict = dict.fromkeys(seq, 10)
print ("New Dictionary : ",  str(dict))
```

```
New Dictionary :  {'name': None, 'age': None, 'sex': None}
New Dictionary :  {'name': 10, 'age': 10, 'sex': 10}
```

Built-in dictionary functions

```
employee = {'Name': 'vema', 'Age': 27};
print('Name: ', employee.get('Name'))
print('Age: ', employee.get('Age'))

# salary key is not vailable and its value by default will be None
print('Salary: ', employee.get('salary'))
print('Salary: ', employee.get('salary',"Not Found ! "))
```

```
Name:  vema
Age:  27
Salary:  None
Salary:  Not Found !
```

```
# get all the keys and its respective values
sales = { 'MacBook Pro': 10, 'iPhone': 132, 'iPad': 78 }
print(sales.items())
```

```
dict_items([('MacBook Pro', 10), ('iPhone', 132), ('iPad', 78)])
```

```
# get only the keys
employee = {'Name': 'vema', 'Age': 27};
print(employee.keys())

empty_dict = {}
print(empty_dict.keys())
```

Built-in dictionary functions

```
employee = {'name': 'Phill'}
print(employee.keys())
print(employee.values())
print(employee.items())
```

```
dict_keys(['name'])
dict_values(['Phill'])
dict_items([('name', 'Phill')])
```

```
# key is not in the dictionary
salary = employee.setdefault('salary' )
print('person = ',employee)
print('salary = ',salary)
```

```
person =  {'name': 'Phill', 'salary': None}
salary =  None
```

```
employee = {'name': 'Phill'}
# key is not in the dictionary, add salarya nd give default value as 5000
salary = employee.setdefault('salary' , 5000)
print('person = ', employee)
print('salary = ', salary)
```

```
person =  {'name': 'Phill', 'salary': 5000}
salary =  5000
```

Built-in dictionary functions

```
# dict1, value of key 2 is three which has to be updated as two from dict2
Dict1 = {1: "one", 2: "three",4:"five"}
print("before update :",Dict1)
Dict2 = {2: "two",4: "four"}


#value of key 2 and 4 from Dict2 is updated with dict1

Dict1.update(Dict2)
print("After update :",Dict1)
```

```
before update : {1: 'one', 2: 'three', 4: 'five'}
After update : {1: 'one', 2: 'two', 4: 'four'}
```

```
# type cast a dict into list
Dict1
list(Dict1)
```

```
[1, 2, 4]
```

Built-in dictionary functions

```
Dict = { 'apple': 2, 'orange': 3, 'grapes': 4 }
print(Dict.values())
```

```
dict_values([2, 3, 4])
```

```
# pass the key and get the value
Dict["apple"]
```

```
2
```

```
#update a dict
Dict["apple"] = 8
Dict
```

```
{'apple': 8, 'orange': 3, 'grapes': 4}
```

```
## delete an object
del Dict["grapes"]
Dict
```

```
{'apple': 8, 'orange': 3}
```

Built-in dictionary functions

```
## each key should be unique
std  = {"Eddy":26,"Eddy":23,"Eddy":28}
print(std) # holds only the last value when the keys are same
```

```
{'Eddy': 28}
```

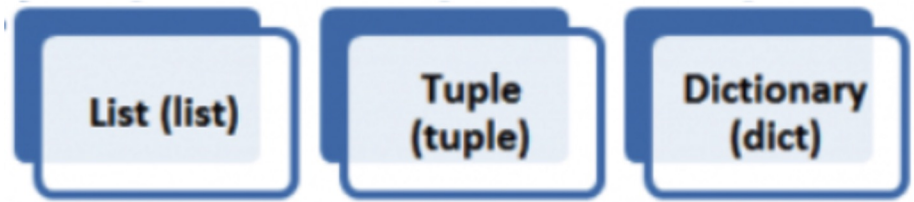
did you know?

Tuples are hashable and lists are not. It means that you can use a tuple as a key in a dictionary. The list can't be used as a key in a dictionary, whereas a tuple can be used

did you know?

If the tuple contains a list or a dictionary inside it, those can be changed even if the tuple itself is immutable.

We learnt list, tuple and dictionary...



	List (list)	Tuple (tuple)	Dictionary (dict)
Create using	[ ]	( )	{ }
Editable	✎	⊘	✎

Set

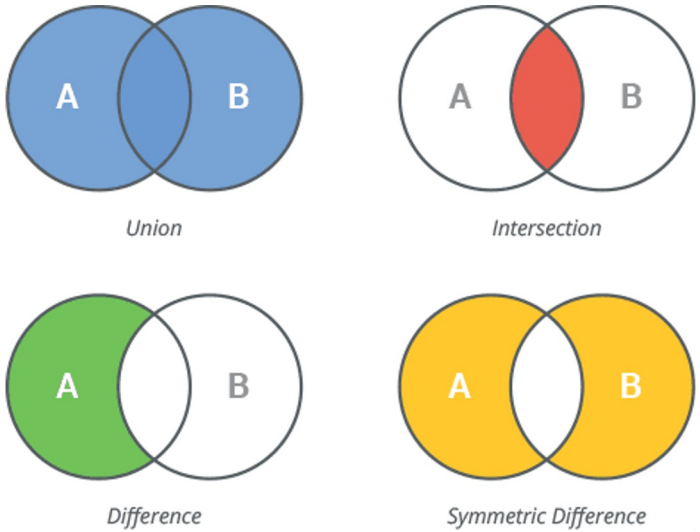
Sets

- Sets are very similar to list data structures
- Do not allow duplicates
- Unordered collections of homogeneous elements
- Sets are generally used to remove duplicate elements from a list

Sets

Sets supports the following operations:

- Intersection
- Union
- Difference
- Symmetric Difference



Sets

```
# Working with sets
# Initialize two sentences.
sentence_1 = "There is nothing new in the world except history you do not know"
sentence_2 = "With the new day comes new strength and new thoughts"
```

```
# Create set of words from strings
sentence_1_words = set(sentence_1.split())
sentence_2_words = set(sentence_2.split())
```

```
# Find out the number of unique words in each set, vocabulary size.
no_words_in_sentence_1 = len(sentence_1_words)
no_words_in_sentence_2 = len(sentence_2_words)
```

```
# Find out the list of common words between the two sets & their count
common_words = sentence_1_words.intersection(sentence_2_words)
number_of_common_words = len(sentence_1_words.intersection(sentence_2_words))
```

```
# Find a list of unique words between the two sets and their count
unique_words = sentence_1_words.union(sentence_2_words)
number_of_unqiue_words = len(sentence_1_words.union(sentence_2_words))
```

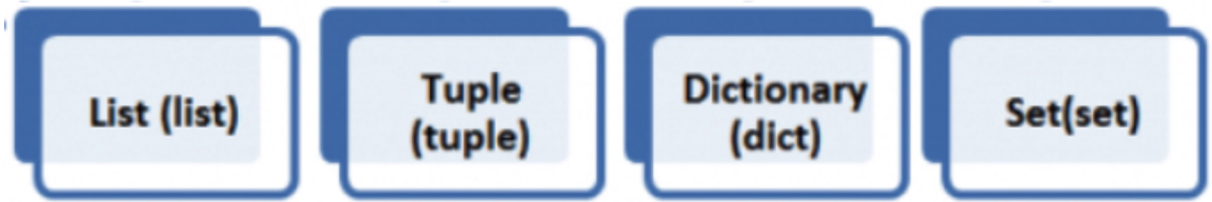
Sets

```
# Find a list of unique words between the two sets and their count
unique_words = sentence_1_words.union(sentence_2_words)
number_of_unqiue_words = len(sentence_1_words.union(sentence_2_words))
```

```
print("Words in sentence 1 = ", sentence_1_words)
print("No of words in sentence 1 = %d"% no_words_in_sentence_1)
print("Words in sentence 2 = ", sentence_2_words)
print("No of words in sentence 2 = %d"% no_words_in_sentence_2)
print("No of words in common = %d"% number_of_common_words)
print("Common words are ", common_words)
print("number of unique words are = %d"%  number_of_unqiue_words)
print("Unique words are ", unique_words)
```

```
Words in sentence 1 =  {'is', 'do', 'not', 'in', 'know', 'nothing', 'the', 'There', 'world', 'you', 'history', 'excep
t', 'new'}
No of words in sentence 1 = 13
Words in sentence 2 =  {'thoughts', 'comes', 'and', 'the', 'With', 'day', 'new', 'strength'}
No of words in sentence 2 = 8
No of words in common = 2
Common words are  {'the', 'new'}
number of unique words are = 19
Unique words are  {'is', 'do', 'not', 'in', 'know', 'comes', 'nothing', 'the', 'There', 'world', 'With', 'history',
'day', 'except', 'thoughts', 'and', 'you', 'new', 'strength'}
```

We learnt the collection objects...



Create using	[ ]	( )	{ }	{ }
Editable	✎	⊘	✎	✎