

# Python Python Everywhere

## ▼ What you can do with Python?

### fields where you can use Python with Libraries & Frameworks

OpenCV	Web Scrapping	Machine Learning	Automation Testing	Web Development	Game Development
* Sciki-image	* Selenium	* Pandas	* PyUnit	* Flask	* Pand3d
* OpenCV	* Beautiful Soup	* Numpy	* PyTest	* Django	* Arcade
* Pillow	* Requests	* Matplotlib	* Behave	* Web2Py	* PyGlet
* Mahotas	* Lxml	* Seaborn	* Robot	* Cheerypy	* PyOpenGL
* SimpleITK	* Scrapy	* Scikit-learn	* Splinter	* Bottle	* PyGame
***	***	* Scipy	***	* Pyramid	***
***	***	* Keras	***	* CubicWeb	***
***	***	* TensorFlow	***	* Dash	***
***	***	* PyTorch	***	* Falcon	***
***	***	* Theano	***	* TurboGears	***

✧ In above table you can see fields like OpenCV, Web Scrapping, Machine Learning, Automation Testing, Web Development, Game Development, these are the all field where you can use Python.

Yes application of Python are huge and wide

### Important Libraries for ML and Data Science

✧ These are the must know libraryies

- **Numpy** - Efficient n-dimensional arrays, Linear Algebra, Random Number capabilities
- **Scipy** - Scientific computing tools like Calculus, Signal Processing
- **Pandas** - Data Reading(multiple formats), manipulation and Cleaning in python
- **Seaborn** - Built on top of Matplotlib, provides a high-level interface for drawing attractive and informative charts
- **Matplotlib** - Fundamental library for Data Visualization
- **StatsModel** - Statistical models, Statical tests, Statical data exploration

- **Scikit-learn** - Basic Data Preprocessing, Machine Learning Library

To learn quick Python I will suggest you to go through:

1. [Basic Python in 1 Hour](#)
2. [Some more Python](#)

As you have gone through above link Lets start learning python basic as your first assignment

## Quick Python Content!

[Hi Code\[1\]](#)

[Variables\(Data Types\)\[2\]](#)

[Assignment Variables & Data Types\[3\]](#)

[Operators\[4\]](#)

[Assignment Operators\[5\]](#)

[Conditional Statements\[6\]](#)

[Assignment Conditional Statements\[7\]](#)

[Loops\[8\]](#)

[Jump Statements\[9\]](#)

[Assignment Loops and Jump Statements\[10\]](#)

[Python Data Structures\[11\]](#)

[Assignment Python Data Structures List\[12\]](#)

[Assignment Python Data Structures List&Set\[13\]](#)

[Functions\[14\]](#)

[Lambda Expressions\[15\]](#)

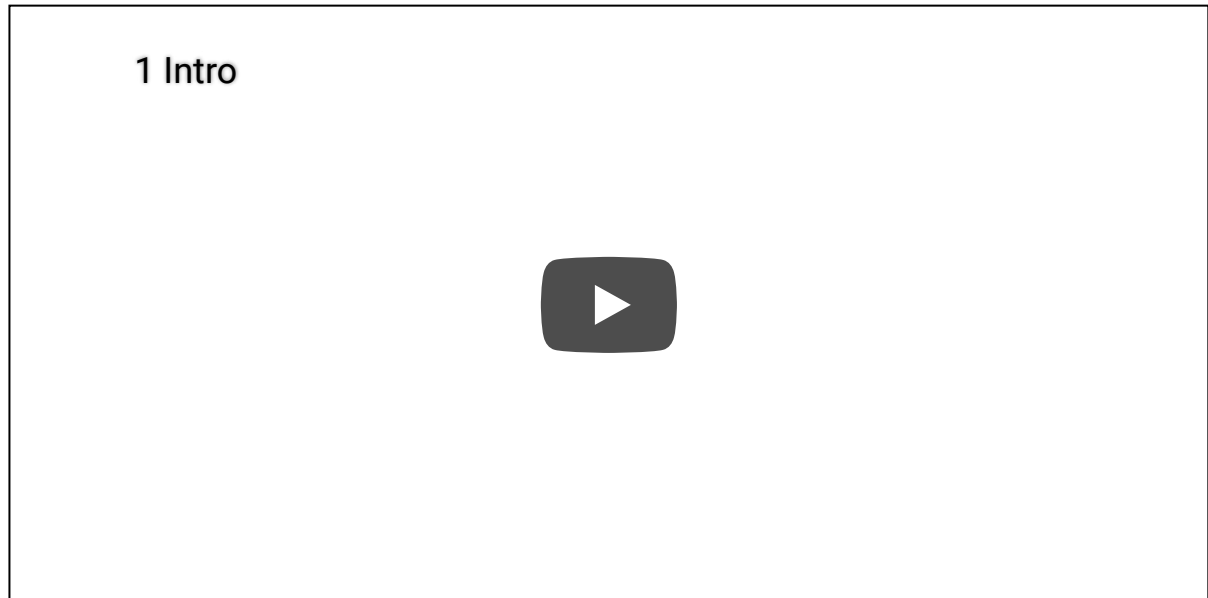
[Comprehensions\[16\]](#)

[Enumerate\[17\]](#)

```
# ignore this code continue please
from IPython.display import HTML
```

Hi, there

YouTubeVideo('https://www.youtube.com/watch?v=izgBmCnp10c&list=PLsR\_0x6BuM-HJZT6Xzj-088N4h



```
# What is your name! Want to print your name!
```

```
Hi [Mukesh Manral] welcome...  
Hi [Mukesh Manral] welcome...
```

- `()` <= Parentheses
- `' '` <= Single Quotes
- `" "` <= Double Quotes
- `\n` <= New\_line
- `#` <= Used to comment inside code

```
# Only use one print function  
# \n <= it will break current line
```

```
Hi [Mukesh Manral] welcome...  
Hi [Mukesh Manral] welcome...
```

Variables

```
# variable-1
```

```
YouTubeVideo('https://www.youtube.com/watch?v=YdDfEbaUPJs&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

**2 variable**



```
# variable-2
```

```
YouTubeVideo('https://www.youtube.com/watch?v=gOcRxdGovhc&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

**3 variable**



```
# define variables named as with values: mukesh 7, z 6, rohan 5, longitude 4,
```

```
# print required variable
```

5

Variable Assignment: **Variable\_Name = Value**

Variables Naming Rules:

- Python is case-sensitive => x=5 is different from X=5 (one is lower and other is upper case)
- var name can't start with special character except underscore(\_) => \_X = 7 is valid, @X = 7 is invalid
- var name can't start with number => 9X = 7 is invalid, X9 = 7 is valid
- can't use keywords as a variable name \*

## ▼ \* Declaring a Variable

```
# declare 4 variables with values as: ur_age 21, ur_weight 50.6, ur_first_name = 'Mukesh', ur_last_name = 'Manral'
```

## ▼ \* Data Type(Type of variable)

Name	Type	Description
Integers	int	Integer number, like 34, -56 ...
Float	float	Decimal number, like 3.4, -5.6 ...
String	str	Ordered sequence of characters, like 'your name'
Boolean	bool	Logical values indicating True or False only

```
# print type of ur_age, ur_weight, ur_first_name, ur_last_name variables
```

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

```
# print values of ur_age, ur_weight, ur_first_name, ur_last_name variables
```

```
21
50.6
Mukesh
Manral
False
```

```
# make 2 variables with values as: ur_first_name 'Mukesh', ur_last_name 'Mukesh'
```

```
# make a variable TrueOrFalse which will have comparison of variables ur_last_name == ur_first_name
```

```
True
```

```
# print variables named as (your name and your age)
```

Name is: Mukesh and age is: 21

## ▼ \*How Variables works in Python

[To see how python code is being executed internally.](#)

```
# quick variables swap
# define 3 variables with values as :x 10, y = x,x 3

# print x,y
```

```
3
10
```

## Assignment Variables & Data Types

# Data Structures

YouTubeVideo('https://www.youtube.com/watch?v=R0yDN07Jw\_c&list=PLsR\_0x6BuM-HJZT6Xzj-088N4h

4 dataStructures



✂ When you create a variable you keep some space in memory. These memory locations are then used to store values

```
# define a variable name "x" and assign value 777 to it
```

- To view some data on screen, python have print function

- Using print function we can control view on output screen

```
# print value of variable x
```

```
777
```

## \*Identifiers in Python

- Like Variables, we have to define Functions , Classes etc in python
  - Naming them helps to differentiate one entity from another
- All these names given to these entities are known as Identifiers .
- Python have some rules to define Identifiers

### **Rules of Writing Identifiers**

- Python is case sensitive language.
- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore \_.
  - Names like myVariable, var\_1 and print\_this, all are valid example.
- An identifier cannot start with a digit
- 1variable is invalid, but variable1 is perfectly fine
- We cannot use special symbols like !, @, #, \$, % etc. in our identifier. \_ is allowed
- **Keywords cannot be used as identifiers => Keywords are the reserved words in Python**

---

### **Keywords in Python**

---

- False
- class
- finally
- is
- return
- None
- continue
- for
- lambda
- try
- True .. etc

## ▼ Type Conversion

***Python defines type conversion functions to directly convert one data type to another.***

```
#convert float to integer
```

```
#1. make a variable named as my_weight_float with datatype float and value 26.33333
#2. make another var named as my_weight_int change above variable datatype to int
#3. print data type of my_weight_float and my_weight_int
#4. print my_weight_float and my_weight_int
```

```
<class 'float'>
<class 'int'>
26.33333
26
```

**+** if we try to add string and integer number, it will give an error

```
# make a variable name with value as: adding_string_and_number '3' + 7
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-34-903f836ab330> in <module>()
----> 1 adding_string_and_number = '3' + 7

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

SEARCH STACK OVERFLOW

```
# convert above string '3' to integer and then add both '3' and 7 and save in variable nam
# called type conversion: string to int
```

```
# print variable adding_string_and_number
```

```
10
```

```
# make two variables with values as: ur_money 18 and frnd_money 21
```

```
# now use f string in print to see how much money you and your friend have
```

```
This is friend money: 21
This is your money: 18
```

```
# swap your money with friend money values
```



```
# now use f string in print to see how much money you and your friend have now after swap
```

```
This is friend money: 18  
This is your money: 21
```

## ▼ \*Python Data Structures

Name	Type	Description
List	list	Ordered sequence of various data types, like [34,'name',3.4]
Dictionary	dict	Unordered sequence of key-values pair, like {'age':34,'age1':56}
Tuples	tup	Non-mutable sequence of data type, like (34,56,'name')
Set	set	Collection of unique elements, like ('id','last_name')

Type	Mutability	Comments
int	immutable	
float	immutable	
string	immutable	
bool	immutable	
frozenset	immutable	immutable version of set
tuple	immutable	immutable version of list
list	mutable	
set	mutable	
dict	mutable	

🔒 immutable => can't be changed

🔓 mutable => can be changed

---

## ▼ Proff that string is immutable

```
# can't reassign
```

```
# make a variable with value as: where_to_learn_MLandDL 'CloudyML'
```

```
# print type of variable
```

```
# access Letter M
```

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

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-38-7a6610e3c481> in <module>()  
# make a variable with value as: where_to_learn_ML and DL 'CloudyMLDL'
```

```
# try to loop in range of (0,10) and try to see memory location of letter-L of string-'Clo
```

```
# If you dont know loop go through videos first then do this
```

```
C = 140430872656560  
l = 140430872679472  
o = 140430872679344  
u = 140430873271600  
d = 140430872678704  
y = 140430872291632  
M = 140430872525104  
L = 140430872525040  
D = 140430872656816  
L = 140430872525040
```

---

```
# list
```

```
YouTubeVideo('https://www.youtube.com/watch?v=VGqMziKr9wM&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

5 list



### List (ordered datastructure)

- Ordered collection of values, each element is separated by a comma, use of square bracket for denotation of list. Ordered property is the most important aspect of list, let's see how:
- Python remembers the order of elements of a list. Data type cannot store different kinds of values in it, but list can.

```
# declare a list named as ur_data_Lis having variables age,weight,name with values as 21,5

# declare a dictionary named as ur_data_dictionary having keys as age,weight, first_name v

# declare a tuple named as ur_data_tuples having values 21,50.6,'first_name'

# declare a set named as ur_data_set having value as Mukesh

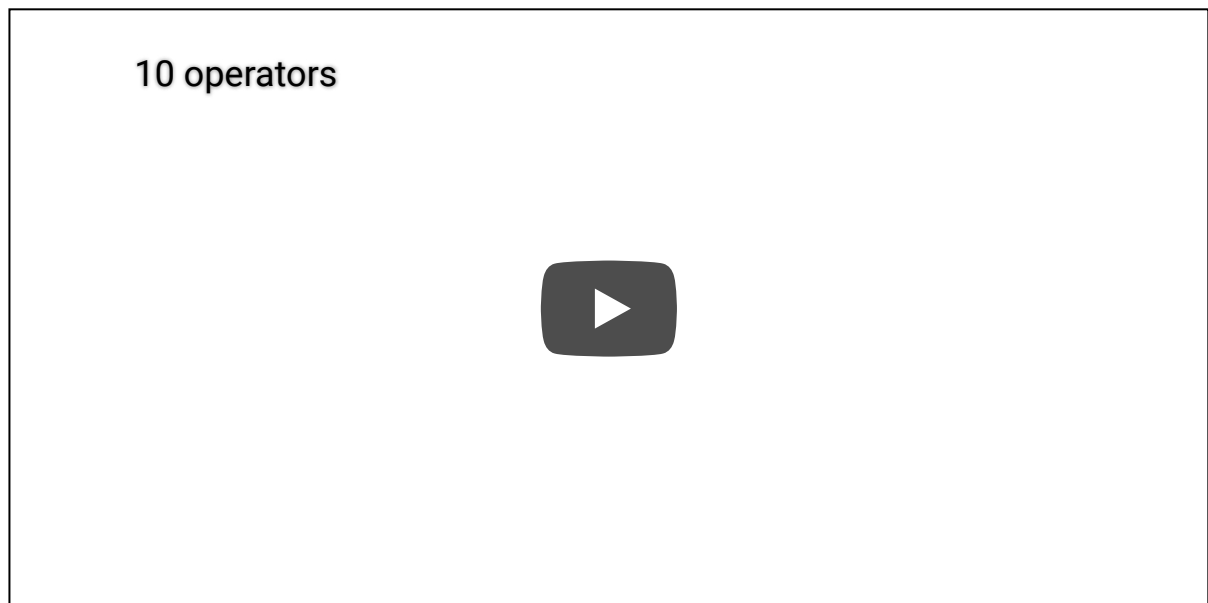
# see datatype of ur_data_List, ur_data_dictionary, ur_data_tuples , ur_data_set variables

<class 'list'>
<class 'dict'>
<class 'tuple'>
<class 'set'>
```

## Operators

# Operators

YouTubeVideo('https://www.youtube.com/watch?v=CLqw5KZZ8Dg&list=PLsR\_0x6BuM-HJZT6Xzj-088N4h



Operators : Symbols that represent mathematical or logical tasks

Example:

700 + 77

- + <= Operator
- 700 & 77 <= Operands

# Types of Operators

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators

## ▼ 1. Arithmetic Operators

Task	Symbol	Example
Addition	+	$700 + 77 = 777$
Substraction	-	$778 - 1 = 777$
Multiplication	*	$1 * 777 = 777$
Division	/	$13 / 2 = 6.5$
Floor Division(ignores decimal \ gives quotient)	//	$13 // 2 = 6$
Modulo(gives remainder)	%	$13 \% 2 = 1$
Exponent(raise to the power)	**	$13^{**2} = 169$

## ▼ 2. Comparison Operators

Task	Symbol
Less than	<
Less than or equall to	<=
Equall to	==
Greater than	>
Greater than or equall to	>=
Not equall to	!=

## ▼ 3. Logical Operators

Task	Symbol	Description
and	$1 < 2 \text{ and } 2 < 3 \Rightarrow \text{True}$	True only if both comparison are true
or	$1 < 2 \text{ or } 2 > 3 \Rightarrow \text{True}$	True if either of the comparison are true
not	$\text{not } 1 > 2 \Rightarrow \text{True}$	(flips results)True if comparison is False and so on..

## Assignment Operators

```
# Initialize variables [x,y,z,zz] with values
## x as 7 =>int ,
## y as 77 =>int,
## z as 77.7 => float,
## zz as 'Hi' => string
```

## ▼ 1. Arithmetic Operators

```
# add x and z
```

```
84.7
```

```
# subtract z and y
```

```
0.70000000000000028
```

```
# Multiply x and z
```

```
543.9
```

```
# Exponent (raise the power or times) x times z
```

```
4.614426248242042e+65
```

```
# division on x and z
```

```
0.09009009009009009
```

```
// => divides and returns integer value of quotient
```

- It will dump digits after decimal

```
# floor division(ignores decimal) on x and z (gives quotient)
```

```
0.0
```

```
# Modulo(gives remainder) on x and z
```

```
7.0
```

## ▼ 2. Comparison Operators

```
# compare and see if x is less than z
# can use '<' symbol
```

```
True
```

```
# check the type of above comparison where it says compare and see if x is less than z
```

```
bool
```

- Bool => takes two values, either True or False

```
# compare and see if x is less than or equal to z
# can use '<=' symbol
```

```
True
```

```
# compare and see if x equal to z
# can use '==' symbol
```

```
False
```

```
# compare and see if x is greater than z
# can use '>' symbol
```

```
False
```

```
# compare and see if x is greater than or equal to z
# can use '>=' symbol
```

```
False
```

```
# compare and see if x is Not equal to z
# can use '!=' symbol
```

```
True
```

### ▼ 3. Logical Operators

```
# compare if 108 is equal to 108, 21 is equal to 21 using logical and
# equal to => '=='
# logical and => and
```

```
# in and both condition must be True to get a True
```

```
True
```

```
# how above condition can give False as output show all those conditions
# in and both condition must be True to get a True
```

```
False
```

```
# compare if 108 is equal to 108, 21 is equal to 11 using logical or
# equal to => '=='
# logical or => or
```

```
# in or Only one condition need to be True to get a True
```

```
True
```

```
# this is for you understand it
(108 == 108) or (21 == 11) or (108 <= 11)
```

```
True
```

## Conditional Statements

```
# Conditions
YouTubeVideo('https://www.youtube.com/watch?v=CLqw5KZZ8Dg&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

## 11 conditions

▼ 1. **if--- else** => to handle single condition

2. **if--- elif--- else** => to handle Multiple condition

Why do we need conditional Statements?

Say you want to buy a ML course and to decided you buy it or not to by it based on some condition,=> if you have more then 2k buy, if you dont have more then 2k dont buy!!

- Two Variable you will use ML course and money
- One comparison `money > 2k` For ML course if `money > 2k` => you have more money then 2k => condition is True => buy course => if condition is False i.e. you have less money then 2k then dont buy the course

Pseudo Code for above will be	Python Code for Pseudo code
Check if money > 2k	if money > 2000:
then ML course = 'buy'	.... ML course = 'buy'
else ML course = 'dont buy'	else:
--	....ML course = 'dont buy'

```
"""
if condition:
    statement
else:
    statement"""

'\nif condition:\n    statement\nelse:\n    statement'
```

Observe in Python code:

- `if` => statement in python
- `else` => statement in python
- `:` => colon => denotes start of if block i.e. any line written after colon belong to if condition
- `....` => see then as indentation i.e. 4 spaces => indentation indicates all code belong to only if and then another indentation indicates code for only else block

```
# make variable with value as : money 100000
```

```
# see output of money > 2000
```

True



```

# see above Python code for Pseudo code and write that into python code

# assign money variable value of 100000
##### say you have this much ammount in your account

# start of if condition
if :# if money is greater then 2000 which is ML course free
    print('buy ML course')
else: # if money > 2000 is false i.e. you have less money then 2000 in your account then e

    buy ML course

# make variable with value as : money 100

# see output of money > 2000

    False

# assign money variable value of 100
#### say you have this much ammount in your account

# start of if condition
if :# if money is greater then 2000 which is ML course free
    print('buy ML course')
else: # if money > 2000 is false i.e. you have less money then 2000 in your account then e

    dont buy ML course

```

## ▼ 2. Multiple Condition i.e. use of if--- elif--- else

Say you have bought ML course, but now you want certification and internship here includes more then 1 condition maybe 2 or 3, lets see for conditions:

- if your test score more then 80 then you will get certification of A grade
- if your test score more then 60 then you will get certification of B grade
- for rest not get anything

### **Pseudo Code for above will be**

---

```

Check if score > 80
if yes print('A grade')
Otherwise check if score > 60
if yes print('B grade')
For every other situation print('not getting anything')

```

# python code for above Pseudo Code will be

```

your_test_score = input('What is your test score(check for 10, 70, 90):') #taking input fr
your_test_score = int(your_test_score) #as input is in string changing it to int as to make comparison possibl

if : # if your_test_score greater then 80
    print('A grade')
elif : # elif your_test_score greater then 60 and your_test_score leass then 80
    print('B grade')
else:
    print('Nothing for you')

What is your test score(check for 10, 70, 90):70
B grade

```

Understand above Flow:

- check for if statement if true close, if not true
- check for elif, if elif condition is true go close, if not true
- go ot else any way
- [See how this code is being executed internally](#)
- copy above code there

## Assignment Conditional Statements

### ✦ ✦ Indentation in Python

***Before we start Conditional Statements, let's have a look at indentation in Python.***

#### INDENTATION

- Indentation means (spaces and tabs) that are used at beginning of any statement
- Statements with same indentation belong to same group also called as suite.

```

# this is how wrong indentation gives you error (commet can be written by using => #)
if (2 == 2):
print('True Statement') # IndentationError

```

```

File "<ipython-input-73-ddff9f6c0e44>", line 3
    print('True Statement') # IndentationError
    ^

```

**IndentationError:** expected an indented block

SEARCH STACK OVERFLOW

Take an input from a user in variable input\_ and print "Even" if number is divisible by 2, otherwise print "Odd"

```
# take input from user by using input function in variable named as input_
```

```
# by default type is string, so we need to convert type first into int
input_ =
```

```
if: # if input_ modulo 2 compared to 0 id true
    print("Even")
else:
    print("Odd")
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-74-8c2d45fec367> in <module>()
      3
      4 # by default type is string, so we need to convert type first
----> 5 input_ = int(input_)
      6
      7 if(input_ % 2 == 0):

ValueError: invalid literal for int() with base 10: ''
```

SEARCH STACK OVERFLOW

Take input from user in variable x and print "Grade A" if y is greater than 90, "Grade B" if y is greater than 60 but less than or equal to 90 and "Grade F" Otherwise

```
# take input from user by using input function
x =
```

```
# by default type is string, so we need to convert type first
x =
```

```
if: # if x is greater then 90
    print("Grade A")
```

```
elif:# elif x is greater then 60
    print("Grade B")
```

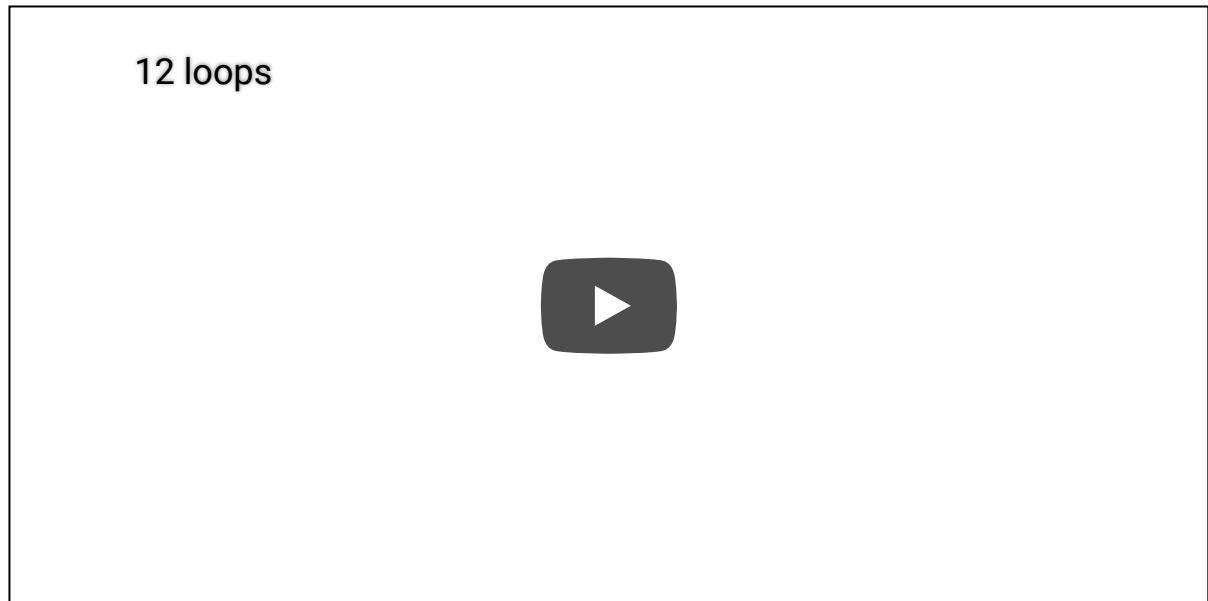
```
else:
    print("Grade F")
```

```
89
Grade B
```

## Loops

## # Loops

YouTubeVideo('https://www.youtube.com/watch?v=56-IOucIwAU&list=PLsR\_0x6BuM-HJZT6Xzj-088N4h



```
"""
```

```
for iterating_variable in sequence:
    statement(s)
```

```
"""
```

```
'\nfor iterating_variable in sequence:\n    statement(s)\n'
```

Loop Type	Description
for in loop	repeats a statement or group of statements finite number of time
while loop	tests condition before executing loop body, repeats a statement or group of statements while a given condition is true
nested loop	loop inside a loop

## When to use while or for loops??

Say your Girlfriend or Boyfriend is mad at you and they asked you to say sorry, but there are two more condition here given by them:

- say sorry 1000 times, now you know end point i.e. 1000 => use for loop
  - say sorry untill I say I love you, now you dont know end point, maybe she/he will never say I love you. => use while loop it will stop saying sorry untill condition is not met
1. We use for loop when we want to run a block of code certain number of times
  2. Code in while clause will be executed as long as while statement's condition is True

### ▼ Why we need Loops?

Say your Girlfriend/Boyfriend is Mad at you and she want you to write sorry 1000 times. What to do now????? Write a for loop

**Pseudo Code for above will be**

---

```
print 'Sorry My love' (repeat 1000 times)
```

```
"""
```

```
for iterating_variable in sequence:
    statement(s)
```

```
"""
```

```
    '\nfor iterating_variable in sequence:\n    statement(s)\n'
```

```
for iterating_variable in range(10):
    print(iterating_variable)
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
# print '🐶 Sorry My Love🐶' 10 times using for loop
```

```
# for now i am giving range of 10 you can give any number you want
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

```
🐶 Sorry My Love🐶
```

10 => stoping criteria of, for loop

```
# Syntax of for loop
```

```
"""
```

```
for iterating_variable in sequence:
    statements(s)
```

```
"""
```

```
'# Syntax of for loop\n\nfor iterating_variable in sequence:\n    statements(s)\n'
```

- in => keyword
- sequence => on which to iterate
- : => colon , start of for loop

#### Pseudo Code while loop

```
while reply != 'I love you':  
    ....keep repeating until  
    ....comparison gives False
```

!= = not equal to => behaves as a stopping criteria

```
# Syntax of while loop  
"""
```

```
while comparison:  
    statements(s)  
"""
```

```
'\nwhile comparison:\n    statements(s)\n'
```

```
# while loop
```

```
# save 0 in variable number  
number =
```

```
# print below result using while loop
```

```
0  
1  
2  
3  
4  
5  
6
```

- Initialized variable number = 0 and then increment its value in each iteration
- Loop will only continue to run only if value is less than 7

## ▼ 3. Nested loop

```
# run it see the output  
for outer_loop_number in range(1,3):  
    print(outer_loop_number)
```

```
1
2
```

```
# run it see the output
for inner_loop_character in "CloudyML":
    print(inner_loop_character)
```

```
C
l
o
u
d
y
M
L
```

```
# print this pattern using outer and inner loop
```

```
1 C
1 l
1 o
1 u
1 d
1 y
1 M
1 L
2 C
2 l
2 o
2 u
2 d
2 y
2 M
2 L
```

See For each step in outer loop, inner loop executes completely

```
# think what will be the output of this code and then run it
# comment why this out is the way it is
```

```
for letter in 'cloudym1':
    if (letter == 'c') or (letter == 'o'):
        print('m', end= ' ')
    else:
        print(letter,end= 'x')
```

```
m lxm uxdyxmxlx
```

Your Comment: => -----

```
# think what will be the output of this code and then run it
```

```
# comment why this out is the way it is
```

```
num = 10
```

```
while num > 5:  
    print(num,end= ' ')
```

```
    num = num - 1
```

```
    if num == 8:
```

```
        num = 2
```

```
10 9
```

Your Comment: => -----

## Jump Statements

```
# Statements-1
```

```
YouTubeVideo('https://www.youtube.com/watch?v=znhhfYiFrzI&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

13 Statementsstory



```
# Statements-2
```

```
YouTubeVideo('https://www.youtube.com/watch?v=znhhfYiFrzI&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```



## 14 Statements application



### ▼ Type of Jump Statements

1. Break Statement
2. Continue Statement
3. Pass Statement

Using for loops and while loops in Python allow you to automate and repeat tasks in an efficient manner.

- But sometimes, an external factor may influence the way your program runs. When this occurs, you may want your program to exit a loop completely, skip part of a loop before continuing, or ignore that external factor

You can do these actions with **Break** and **Continue** Statements

```
'''
pass

break

continue
'''

'\npass\n\nbreak\n\ncontinue\n'
```

### ▼ 1. Break Statement

```
# observe the output
for num in range(10):
    print(num)
```

```
0
1
2
3
4
5
6
7
```

8  
9

# example that uses break statement in a for loop

```
num = 0
for num in range(10):
    if num == 5:
        pass#break when num == 5
    print('The number is' + str(num))

print('Out of loop')
```

# Explain Why the output is the way it is

```
The number is0
The number is1
The number is2
The number is3
The number is4
The number is5
The number is6
The number is7
The number is8
The number is9
Out of loop
```

Your Comment: => -----

## ▼ 2. Continue Statement

# Using same `for loop program` as in Break Statement section above, we'll use a continue  
num = 0

```
for num in range(10):
    if num == 5:
        continue #keep running if num == 5
    print('The number is' + str(num))

print('Out of loop')
```

# Explain Why the output is the way it is

```
The number is0
The number is1
The number is2
The number is3
The number is4
The number is6
```

```
The number is7
The number is8
The number is9
Out of loop
```

Your Comment: => -----

```
# Using same code block as above, let's replace break or continue statement with a pass st
num = 0
```

```
for num in range(10):
    if num == 5:
        pass    # pass then
    print('Number is ' + str(num))
print('Out of loop')
```

```
# Explain Why the output is the way it is
```

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Number is 5
Number is 6
Number is 7
Number is 8
Number is 9
Out of loop
```

Your Comment: => -----

## Assignment Loops and Jump Statements

### ▼ **Problem Statement-1:**

- User will enter password which is set to be `qwert`
- If password matches, print *Welcome to CloudyML Course!!* else print *Wrong Password!!* and user will re-enter password
- In fourth trial, it will print *One Last Trial Left!!!*

Hint: Break Statement can be used

```
password = "qwert"
```

```
Enter Password:
Wrong Password!!
Enter Password: qwert
Welcome to CloudyML Course!!
```

.....

## ▼ **Problem Statement-2:**

- Enter password which is an OTP set to be 1008
- If OTP matches, It will ask user to provide details like *Enter Your Name:* and *Enter Your city:* and user will re-enter OTP
- In fourth trial, it will print *One Last Trial Left!!!*

Hint: Continue Statement can be used

```
OTP = "1008"
```

```
Enter OTP: 1008
Enter Your Name: Mukesh Manral
Enter Your City: Gibrish
```

## Python Data Structures

```
# Data Structures
```

```
YouTubeVideo('https://www.youtube.com/watch?v=R0yDN07Jw_c&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

4 dataStructures



Existing Data Type: int, float, bool, str

Problem with Existing Data Type:

- Only single data type can be stored in a variable i.e. either int or decimal or str
- To store more data we need more variables
  - Eg: You have 100 name and age then you have to make 200 variables, valid for condition if you are not using DataStructure

Data Structure Solves above problem

## ▼ Properties of Data Structure:

- Efficient Storage for large data
- Should allow Manipulation/Operations on data
- Should maintain underlying relationship of data If these three properties are available then you can call it a Data Structure

## Types of Data Structures:

- List => ordered => elements separated by comma => enclosed with square brackets-[]
- Tuple
- Set
- Dictionary

Data Structures type	Mutable	Comments	Indexing	Ordered	Duplicacy
frozenset	immutable	immutable version of set	-	-	-
tuple ()	immutable	immutable version of list	possible	yes	allowed
list []	mutable	-	possible	yes	allowed
set {}	mutable	-	not	no	not
dict {key:value}	mutable	-	possible	no	not

🔒 immutable => can't be changed

🔓 mutable => can be changed

---

# List

YouTubeVideo('https://www.youtube.com/watch?v=VGqMziKr9wM&list=PLsR\_0x6BuM-HJZT6Xzj-088N4h

## 5 list



### ▼ List

```
# make a list, name it as names with values ram','ohm','da','ada','asdasd'

# make a list, name it as roll_no with values 21,23,435,65,565,656

# make a list, name it as scores with values 23,2323,2323,234324,3423


# make a list with Single data type - str
# name of list names_list with values as 'krishna','mahadev','surya'

# print type of names_list

list

# make a list with Mixed data type - str and int
# name of list names_list with values as 'krishna','mahadev','surya',108,11,21

# print type of names_list

list

# access output element

'mahadev'
```

List Indexing:

Values	1	2	3	4	5	6	7	8	9	10
Indexing	0	1	2	3	4	5	6	7	8	9
Negative Indexing	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
# extract single element i.e. last one using Negative Index (first element)
```

```
# extract single element using Index (first element)
```

```
'krishna'
```

```
# extract sequence of elements using Index (first 3 elements)
```

```
#=> [start index:end index] => exclude end index
```

```
['krishna', 'mahadev', 'surya']
```

## ▼ Membership Operators

To check if particular element is present in list or not

- Using `in` or `not in` to get answers of these type of queries

```
# see names_list data
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21]
```

```
# see if 'mukesh' is in names_list
```

```
False
```

```
# see if 'rawan' is not in names_list
```

```
True
```

## Adding to List

- Adding a single element: `append()` => add at the end of the list
- Adding multiple elements (adding list in list): `extend()`

## ▼ Use of `.append()`

```
#see again names_list elements
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21]
```

```
# #see again names_list list elements
```

```
#appending 'bhagwan' to list names_list  
#using append()
```

```
# print list after appending
```

```
# append 3,'ok' values as a list of list  
#<= it will append a list inside a list
```

```
# print names_list
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21]
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan']
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan', [3, 'ok']]
```

## ▼ Use of .extend()

```
# extend names_list with values 6,'one more'
```

```
# print list after .extend()
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan', [3, 'ok'], 6, 'one more']
```

## Deleting from list

- .remove() => delete element by its value
- del => delete element by index : indexing start from 0

## ▼ Use of .remove()

```
# see names_list
```

```
['krishna',  
 'mahadev',  
 'surya',  
 108,  
 11,  
 21,
```



```
'bhagwan',  
[3, 'ok'],  
6,  
'one more']
```

```
# remove one more from names_list
```

```
# print names_list
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan', [3, 'ok'], 6]
```

## ▼ Use of del

```
# see names_list
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan', [3, 'ok'], 6]
```

```
# del 7 from names_list
```

```
# print names_list
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan', 6]
```

## ▼ Iterate through list

```
# iterate through list elements
```

```
# print element
```

```
krishna  
mahadev  
surya  
108  
11  
21  
bhagwan  
6
```

## ▼ Some pre-defined method on List

```
# see names_list
```

```
['krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan', 6]
```

```
# find length of list names_list
```

```
# reverse a list names_list
# see names_list
```

```
[6, 'bhagwan', 21, 11, 108, 'surya', 'mahadev', 'krishna']
```

```
# sort a list see condition when it is not possible
names_list.sort()
names_list
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-123-4d979fc41e2f> in <module>()
      1 # sort a list
----> 2 names_list.sort()
      3 names_list

TypeError: '<' not supported between instances of 'str' and 'int'
```

SEARCH STACK OVERFLOW

```
# make a list named as numbers_list with values as [51,101,11,21]
# sort list numbers_list
# see numbers_list output
```

```
# another way of sorting list
```

```
# use sorted to sort list named as sample_list with values as [51,101,11,21]
```

```
[11, 21, 51, 101]
```

## ▼ Nested List

```
# defining a sample nested list named as list_of_list and values [111, 201, [11, 21, 121]]
```

```
# print list_of_list
```

```
[111, 201, [11, 21, 121], ['Aman', 'Bobo', 'Canada'], 'Wangu', ['Aanaya', 'Annu', 'Anu']]
```

```
# find length of nested list
```

```
# access third element of list_of_list
```

```
[11, 21, 121]
```

```
# accessing 2nd element of the third element of the primary list named as list_of_list
```

21

## Assignment Python Data Structures List

### ▼ List Problem Statement:

Say CloudyML conducted an assessment test to hire a Data Scientist

Candidates were evaluated on 5 different subject Statistics, Python, SQL, ML, and DeepLearning

Help CloudyML team to find out answers of following questions

💡 Find out Who scored highest marks in Python?

- Step1: Select name of student and marks in Python
- Step2: Store filtered name and marks in another list
- Step3: Sort list. (Kept marks as first index)
- Step4: Get final index of sorted list

```
#### Marks of 5 different subjects out of 100 are as ####
```

```
# make student marks nested list
student_marks =
```

```
# print student_marks
```

```
[['Name', ['Statistics', 'Python', 'SQL', 'ML', 'DeepLearning']],
 ['Ananya', [41, 34, 45, 55, 63]],
 ['Akash', [42, 23, 34, 44, 53]],
 ['Rahul', [32, 23, 13, 54, 67]],
 ['Mukesh', [23, 82, 23, 63, 34]],
 ['Pranav', [21, 23, 25, 56, 56]]]
```

```
# access first index of list student_marks
```

```
['Name', ['Statistics', 'Python', 'SQL', 'ML', 'DeepLearning']]
```

- Slice list from second element

```
# access first index of list student_marks[1]
```

```
['Ananya', [41, 34, 45, 55, 63]]
```

See, each element in list from Index-1 contains 2 elements

- first is Name
- second is list of marks

```
# access first student name
```

```
'Ananya'
```

```
# access ananya marks
```

```
[41, 34, 45, 55, 63]
```

Now, student marks are in order **Statistics**, **Python**, **SQL**, **ML** and **DeepLearning**. and we need to access value at Index-1 to get marks in subject => **Python**

```
# observe the output comment why this is the way it is  
student_marks[1][1][1]
```

```
34
```

See above Ananya Python marks

### ***Now Solving Problem step after step***

- Who scored highest marks in subject Python?

#### ***Step1: Select name of student and marks in subject Python***

```
Ananya 34  
Akash 23  
Rahul 23  
Mukesh 82  
Pranav 23
```

#### ***Step2: Store filtered Name and Marks in another list***

```
[[34, 'Ananya'], [23, 'Akash'], [23, 'Rahul'], [82, 'Mukesh'], [23, 'Pranav']]
```

### Step3: Sort list (Keep marks as first index)

```
[[23, 'Akash'], [23, 'Pranav'], [23, 'Rahul'], [34, 'Ananya'], [82, 'Mukesh']]
```

### Step4: Get final index of sorted list with student name having max marks in Python subject

```
'Mukesh'
```

Got answer of first question. Mukesh has scored highest marks in subject Python

- You can also do it using loop do it if you want to

---

# Tuples

YouTubeVideo(' [https://www.youtube.com/watch?v=I6UH\\_prRboE&list=PLsR\\_0x6BuM-HJZT6Xzj-088N4h](https://www.youtube.com/watch?v=I6UH_prRboE&list=PLsR_0x6BuM-HJZT6Xzj-088N4h)')

6 tuples



## ▼ Tuple

Benefit of using tuples:

- Faster than list
- Provide security over updation => once created cannot be modified like list
- Can be used as key for dictionary

- We generally use tuple for heterogeneous(different) datatypes and list for homogeneous (similar) datatypes
- Since tuples are immutable, iterating through tuple is faster than with list. So there is a slight performance boost
- Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected
- We cannot edit/update or delete elements from the tuple

```
# make a variable names as data_tuple with values 'krishna', 'mahadev', 'surya', 108, 11,
# print data_tuple

# print type of data_tuple

('krishna', 'mahadev', 'surya', 108, 11, 21, 'bhagwan', 6)
<class 'tuple'>

# make a variable named as data_tuple with values as 1,2,3,4,5
# Access element at 2nd Index of data_tuple  #<= indexing work same as list as tuple is ord

3

# think what will be the output of this code then run it
data_tuple = ( (1,2,3,4,5),
               (4,5,3,4,5),
               (3,7,8,9,6) )

data_tuple[2][2:4]

# Views on why the output is the way it is

(8, 9)

# find Index of any element, here index of 3
data_tuple = (1,2,3,4,5)
# <= 3 is at 2nd index as indexing start at one

2

# count of an element
data_tuple = (1,2,3,4,5)
# print how many time 4 have come
```

```
data_tuple = (7,2,7,4,7)
# print how many time 7 have come
```

```
1
3
```

```
# iterate through elements of tuple similar to list using for
data_tuple = (7,2,7,4,7)
```

```
7
2
7
4
7
```

```
# slicing in tuple
data_tuple = (5,2,7,4,6)
```

```
# slice and print first 2 element
#<= exclude 2nd Index
```

```
# defining start point as 2 and end point as 4 to slice
```

```
# by using count method on slice see how many time 2 comes
#<= count of 2 if present
```

```
# index method on slice
print( data_tuple[1:4].index(2) ) #<= index of 2
```

```
(5, 2)
(7, 4)
0
0
```

```
# Tuple immutable (cant update or delete elements from tuples)
## this is one of the reasons it is fast in comparison to list
data_tuple[2] = 1 #<= update at index-2 with 1, it will give error
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-149-dc310a939966> in <module>()
      1 # Tuple immutable (cant update or delete elements from tuples)
      2 ## this is one of the reasons it is fast in comparison to list
----> 3 data_tuple[2] = 1 #<= update at index-2 with 1, it will give error
```

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

SEARCH STACK OVERFLOW

```
data_tuple
```

```
(5, 2, 7, 4, 6)
```

```
del data_tuple[0] #<= del at index-0, again error
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-151-80fde943550a> in <module>()  
----> 1 del data_tuple[0] #<= del at index-0, again error  
  
TypeError: 'tuple' object doesn't support item deletion
```

SEARCH STACK OVERFLOW

## Assignment Python Data Structures List&Set

### ▼ List Problem Statement:

Say CloudyML conducted an assessment test to hire a Data Scientist

Candidates were evaluated on 5 different subject Statistics, Python, SQL, ML, and DeepLearning

Help CloudyML team to find out answers of following questions

- 💡 Find out Who scored highest marks in Python?
- 💡 Find out average marks scored in subject SQL?
- 💡 Find out scored highest percentage of marks?
- 💡 If considered only top-4 subjects of a candidate, then who scored highest percentage of marks?

```
### Find out Who scored highest marks in Python? ###
```

```
# student marks nested list
```

```
student_marks = [['Name', ['Statistics', 'Python', 'SQL', 'ML', 'DeepLearning']],  
                 ['Ananya', [41, 34, 45, 55, 63]],  
                 ['Akash', [42, 23, 34, 44, 53]],  
                 ['Rahul', [32, 23, 13, 54, 67]],  
                 ['Gyan', [23, 82, 23, 63, 34]],  
                 ['Pranav', [21, 23, 25, 56, 56]] ]
```

```
# Find out Who scored highest marks in Python? using for loop, sorted and slicing
```

```
[82, 'Gyan']
```



### Find out average marks scored in subject SQL? ###

```
# Extracted col= name with SQL_data=marks
```

```
# Extracted only SQL_data=marks
```

```
# mathematical operation
```

```
avg_of_SQL_marks = sum(all_SQL_marks)/total_element_in_SQL
```

```
print('avg_of_SQL_marks:',avg_of_SQL_marks)
```

```
avg_of_SQL_marks: 28.0
```

### Find out scored highest percentage of marks? ###

```
Maximum marks for each subject:70
```

```
Number of Subjects:5
```

```
Topper percentage is: 48 % Name is: Ananya
```

### solv for `If considered only top-4 subjects of a candidate, then who scored the highest

```
Maximum marks for each subject:67
```

```
Number of Subjects:5
```

```
Topper percentage is: 41 % Name is: Ananya
```

---

```
# Sets
```

```
YouTubeVideo('https://www.youtube.com/watch?v=I6UH_prRboE&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

**8 set**



## ▼ Sets

- Unordered collection of elements (if you put some data in order inside set it will discard order)
  - It sorts elements by itself (see below example)
- No duplicates only unique values
  - Put duplicates in set it will auto remove them
- Mutable can add, remove values
- Set Theory operations are possible like Union, Intersection etc...
  - intersection of 2 sets => set\_1.intersection(set\_2)
  - union of 2 sets => set\_1.union(set\_2) ..

```
# define set named as set_ with values 'b','c','a','z'
```

```
# print set
```

```
{'a', 'b', 'c', 'z'}
```

```
# define list named as frt with elements as 'mango','banana','mango','banana'
```

```
# define a set named as set_ with elements as 'mango','banana','mango','banana'
```

```
# print set
```

```
{'banana', 'mango'}
```

```
# make a set named as set_data and values as 4,'A',5,1,'V','L'
```

```
# print set_data #<= results are sorted
```

```
{1, 4, 5, 'A', 'L', 'V'}
```

```
# No duplicates only unique values
```

```
# make a set named as set_data and values as 4,4,5,1,'V','V'
```

```
# print set_data
```

```
{1, 4, 5, 'V'}
```

```
# make tuple named as tuple_data with values as 'A', 'B', 1, 2, 'B', 'A'
```

```
# make a variable names as var_1 and convert it into set
```

```
# print var_1
```

```
{1, 2, 'A', 'B'}
```

```
# Defining a set using a List
```

```
# define a list named as list_data with values as 'A', 'B', 1, 2, 'B', 'A'] # this is a l
```

```
# change set_from_list to set # set will remove duplicates
```

```
# print set_from_list)
```

```
# Defining a set using a Tuple named as tuple_data with values as 'A', 'B', 1, 2, 'B', 'A'
```

```
# make variable named as set_from_tuple, change tuple_data into set
```

```
# print set_from_tuple)
```

```
{'A', 2, 1, 'B'}
```

```
{'A', 2, 1, 'B'}
```

```
# see set_from_tuple output
```

```
{1, 2, 'A', 'B'}
```

```
# add 0 to set_from_tuple
```

```
# print set_from_tuple
```

```
{0, 1, 2, 'A', 'B'}
```

```
# make variable named as set_ with values as 'b','c','a','z'
```

```
# print set_
```

```
{'a', 'b', 'c', 'z'}
```

```
# add x to set_
```

```
# print set_
```

```
{'a', 'b', 'c', 'x', 'z'}
```

```
# Add 0 to set set_from_tuple
```

```
# print set_from_tuple # 0 is added at the starting of set, as set orders by itself
```

```
# adding C to set set_from_tuple
```

```
# print set_from_tuple # c is added at the last of set, as set orders by itself
```

```
{0, 1, 2, 'A', 'B'}
```

```
{0, 1, 2, 'A', 'B', 'C'}
```

## ▼ delete element from set

```
# see output of set_from_tuple

{0, 1, 2, 'A', 'B', 'C'}

# use discard to remove 0 from set_from_tuple
# see output of set_from_tuple

{1, 2, 'A', 'B', 'C'}

# from set_from_tuple discard 9
# print set_from_tuple

{1, 2, 'A', 'B', 'C'}

# delete element from set (two methods are as)
## discard() : if element not in set, it will not show error
## remove() : if element not in set, it will show error
set_from_tuple.discard(0.0)
```

## ▼ Difference b/w two set (set\_1 - set\_2)

```
# set_1 - set_2 => states that, collect every element of set_1 after removing those elements

# define a set named as set_1 with elements as 1,2,3,'A','B'
# define a set named as set_2 with elements as 3,'A'
# difference of set_1 and set_2
# see difference between set_1 and set_2)
# print set_1

{1, 2, 3, 'A', 'B'}

# if you want to update set_1 by difference

# using difference_update update set_1 and set_2
# print set_1

{1, 2, 'B'}
```

## ▼ Intersection b/w two set (set\_1 - set\_2)

```
# collect only elements which are common b/w both sets
set_1 = {1,2,3,'A','B'}
set_2 = {3,'A'}
set_1.intersection(set_2)
set_1

{1, 2, 3, 'A', 'B'}
```

```
# if you want ot update set_1 by intersection
set_1.intersection_update(set_2)
set_1

{3, 'A'}
```

---

```
# Dict
YouTubeVideo('https://www.youtube.com/watch?v=I6UH_prRboE&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```



## ▼ Dictionary => key:value

- Unordered data structure
- Collection of multiple lists can say, looks like set{} but have {key:value} pairs for storing elements
  - student\_information is a dictionary and it have name,age,email,adress etc store in form of list how see bellow example
- Element of dictionary can only be accessed by using key not Index
- Every key must be unique

{ } -> curly brac

```
# student details
names = ['Viranda', 'Sandhu', 'Rama', 'Yoga']
height = [120,130,140,150]
weight = [56,60,45,90]
age = [23,34,56,55]
```

```
"""
```

```
dict = { key:values }
```

```
"""
```

```
### storing all these 4 lists in a dict using key:value # once data structure inside othe
```

```
student_details_dict = {  
    'names' : ['Viranda', 'Sandhu', 'Rama', 'Yoga'],  
    'height' : [120,130,140,150],  
    'weight' : [56,60,45,90],  
    'age' : [23,34,56,55]  
}
```

- Above name,height are the key and list's data are the value pair associated with the key
  - can say each key is mapped to multiple values
- Above is the representation of advanced data structure as you can see list inside a dictionary

```
### Accessing elements of dict ###
```

```
student_details_dict = {  
    'names' : ['Ohm', 'Sandhu', 'Rama', 'Yoga'],  
    'height' : [120,130,140,150],  
    'weight' : [56,60,45,90],  
    'age' : [23,34,56,55]  
}
```

```
# trying to access using index will give error  
student_details_dict[0]
```

```
-----  
KeyError                                Traceback (most recent call last)  
<ipython-input-181-939d01fd8d6e> in <module>()  
      8  
      9 # trying to access using index will give error  
>>> 10 student_details_dict[0]  
  
KeyError: 0
```

SEARCH STACK OVERFLOW

```
student_details_dict['age']
```

```
student_details_dict
```

```
{'age': [23, 34, 56, 55],  
 'height': [120, 130, 140, 150],  
 'names': ['Ohm', 'Sandhu', 'Rama', 'Yoga'],  
 'weight': [56, 60, 45, 90]}
```

```
student_details_dict['names'][-1]
```

```

        'Yoga'
# trying to access using key will give error

print(student_details_dict['names'], '\n')

# getting into name key values
print(student_details_dict['names'][1:])

        ['Ohm', 'Sandhu', 'Rama', 'Yoga']

        ['Sandhu', 'Rama', 'Yoga']

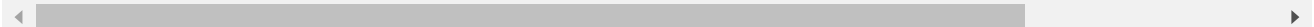
### Accessing keys ###
student_details_dict.keys()

        dict_keys(['names', 'height', 'weight', 'age'])

### Accessing values ###
student_details_dict.values()

        dict_values(['Ohm', 'Sandhu', 'Rama', 'Yoga'], [120, 130, 140, 150], [56, 60, 45, 90])

```



## ▼ Adding single element to dictionary

```

student_details_dict

{'age': [23, 34, 56, 55],
 'height': [120, 130, 140, 150],
 'names': ['Ohm', 'Sandhu', 'Rama', 'Yoga'],
 'weight': [56, 60, 45, 90]}

# dict_name['key_name'] = [values to update]
student_details_dict['emails'] = ['as@gmail.com', 'asd@gmail.com', 'd@gmail.com', 'sd@gmail.c

student_details_dict

{'age': [23, 34, 56, 55],
 'emails': ['as@gmail.com', 'asd@gmail.com', 'd@gmail.com', 'sd@gmail.com'],
 'height': [120, 130, 140, 150],
 'names': ['Ohm', 'Sandhu', 'Rama', 'Yoga'],
 'weight': [56, 60, 45, 90]}

```

## ▼ Adding multiple element to dictionary

```

student_details_dict

```

```
{'age': [23, 34, 56, 55],
 'emails': ['as@gmail.com', 'asd@gmail.com', 'd@gmail.com', 'sd@gmail.com'],
 'height': [120, 130, 140, 150],
 'names': ['Ohm', 'Sandhu', 'Rama', 'Yoga'],
 'weight': [56, 60, 45, 90]}
```

```
# use of .update({'key':[value],'key1':[value1]})
```

```
student_details_dict.update({'adress':['ana','dasd','asdad','adasd'],'sex':['m','m','m','m']})
student_details_dict
```

```
{'adress': ['ana', 'dasd', 'asdad', 'adasd'],
 'age': [23, 34, 56, 55],
 'emails': ['as@gmail.com', 'asd@gmail.com', 'd@gmail.com', 'sd@gmail.com'],
 'height': [120, 130, 140, 150],
 'names': ['Ohm', 'Sandhu', 'Rama', 'Yoga'],
 'sex': ['m', 'm', 'm', 'm'],
 'weight': [56, 60, 45, 90]}
```

## ▼ Deleting element of dictionary

```
student_details_dict
```

```
{'adress': ['ana', 'dasd', 'asdad', 'adasd'],
 'age': [23, 34, 56, 55],
 'emails': ['as@gmail.com', 'asd@gmail.com', 'd@gmail.com', 'sd@gmail.com'],
 'height': [120, 130, 140, 150],
 'names': ['Ohm', 'Sandhu', 'Rama', 'Yoga'],
 'sex': ['m', 'm', 'm', 'm'],
 'weight': [56, 60, 45, 90]}
```

```
# use of del
```

```
del student_details_dict['sex'] #<= removing sex key totally from key
student_details_dict
```

```
{'adress': ['ana', 'dasd', 'asdad', 'adasd'],
 'age': [23, 34, 56, 55],
 'emails': ['as@gmail.com', 'asd@gmail.com', 'd@gmail.com', 'sd@gmail.com'],
 'height': [120, 130, 140, 150],
 'names': ['Ohm', 'Sandhu', 'Rama', 'Yoga'],
 'weight': [56, 60, 45, 90]}
```

## ▼ Updating dict with dicts

```
# Make a dict named as student_data_dictionary with value which are given bellow as output
student_data_dictionary = {}
```

```
# print student_data_dictionary
```



```
{'age': 21,
 'attendance': 74,
 'city': 'Gurugram',
 'country': 'India',
 'name': 'Vany',
 'percentage': 85,
 'roll_no': 'CSE100'}
```

```
# update student_data_dictionary by observing output
```

```
# print student_data_dictionary
```

```
{'age': 21,
 'attendance': 74,
 'city': 'Gurugram',
 'country': 'India',
 'last_name': 'Unicorn',
 'name': 'Vany',
 'percentage': 85,
 'pincode': 345467,
 'roll_no': 'CSE100'}
```

## ▼ Updating dict with list of tuples

```
# print student_data_dictionary
```

```
{'age': 21,
 'attendance': 74,
 'city': 'Gurugram',
 'country': 'India',
 'last_name': 'Unicorn',
 'name': 'Vany',
 'percentage': 85,
 'pincode': 345467,
 'roll_no': 'CSE100'}
```

```
# update student_data_dictionary by observing output
```

```
# print student_data_dictionary
```

```
{'age': 21,
 'attendance': 74,
 'city': 'Gurugram',
 'country': 'India',
 'last_name': 'Unicorn_777',
 'name': 'Vany',
 'percentage': 85,
 'pincode': 423423,
 'roll_no': 'CSE100'}
```

```
# observe this code
```

```
### dictionary key can be anything string,integer,float,boolean ###
```

```
dict_sample = {
```

```

    'string' : 'string can be used as key',
    777 : 'integer can be used as key',
    777.77 : 'float can be used as key',
    False : 'boolean can be used as key'
}

dict_sample.keys()

dict_keys(['string', 777, 777.77, False])

```

Mostly String and Integer are used as key

```

dict_sample[777.77]

'float can be used as key'

```

```

# observe this code
### Duplicate keys ###
# every key must be unique
dict_sample = {
    'a':1,
    'b':2,
    'c':3,
    'a':23
}

```

```

dict_sample

{'a': 23, 'b': 2, 'c': 3}

```

If we add duplicate keys, it will not throw an error instead it will overwrite value which is added at last

## ▼ Iterating through key:value pair

```

dict_sample.items()

dict_items([('a', 23), ('b', 2), ('c', 3)])

for key,value in dict_sample.items():
    print(key,value)

a 23
b 2
c 3

for key,value in dict_sample:

```

```
print(key,value)
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-202-ccb80a8915b3> in <module>()  
----> 1 for key,value in dict_sample:  
      2     print(key,value)
```

**ValueError:** not enough values to unpack (expected 2, got 1)

SEARCH STACK OVERFLOW

## ▼ sorting a dictionary based on keys

```
sorted(dict_sample.items())
```

```
[('a', 23), ('b', 2), ('c', 3)]
```

```
# sorting a dictionary based on values  
# it requires concept of lambda expresiion
```

---

```
# String Manipulation  
YouTubeVideo('https://www.youtube.com/watch?v=I6UH_prRboE&list=PLsR_0x6BuM-HJZT6Xzj-088N4h')
```

9 string manipulation



String Manipulation [More Resources](#)

## ▼ observe bellow code

```
string_ = ' ' or '"' or '""' or ''''
```

```
# defining a string
```

```
string = "We are creating next generation data science eco-system at CloudyML"  
string
```

```
'We are creating next generation data science eco-system at CloudyML'
```

```
# length of string including spaces
```

```
len(string)
```

```
67
```

```
# Accessing characters in a string
```

```
string[0]
```

```
'W'
```

```
# Accessing characters with negative indexing
```

```
string[-1]
```

```
'L'
```

## ▼ String Slicing

```
string[:6]
```

```
'We are'
```

```
string[7:-10]
```

```
'creating next generation data science eco-system a'
```

## ▼ Count of a particular character in a string

```
print(string)
```

```
string.count('e')
```

```
We are creating next generation data science eco-system at CloudyML  
10
```

## ▼ Count of a particular sub-string in a string

```
print(string[7:-10])
string.count('ea')
```

```
creating next generation data science eco-system a
1
```

## ▼ Find a substring in string using find and index function

```
string
```

```
'We are creating next generation data science eco-system at CloudyML'
```

```
# .find() => if present it will return starting index, not found then it will return -1
# .index() => if present it will return starting index, not found then it will give error
```

```
string.find('eaq')
```

```
-1
```

```
string.index('eaq')
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-219-d507e4bda74d> in <module>()
----> 1 string.index('eaq')
```

```
ValueError: substring not found
```

SEARCH STACK OVERFLOW

```
string
```

```
'We are creating next generation data science eco-system at CloudyML'
```

```
### Checking whether string `startswith` or `endswith` a particular substring or not ###
string.endswith('CloudyML')
```

```
True
```

```
string.startswith('We')
```

```
True
```

```
string
```

```
'We are creating next generation data science eco-system at CloudyML '
```

```
### Converting string to upper case ###
```

```
string.upper()
```

```
'WE ARE CREATING NEXT GENERATION DATA SCIENCE ECO-SYSTEM AT CLOUDYML '
```

```
### Converting only first character of string to upper case ###
```

```
string.capitalize()
```

```
'We are creating next generation data science eco-system at cloudyml '
```

```
string
```

```
'We are creating next generation data science eco-system at CloudyML '
```

```
### Checking if string is in lower case or upper case ###
```

```
string.islower()
```

```
False
```

```
string.isupper()
```

```
False
```

```
### Checking if string is digit, alphabetic, alpha-numeric ###
```

```
"1".isnumeric()
```

```
True
```

```
"1s".isnumeric()
```

```
False
```

```
"name".isalpha()
```

```
True
```

```
"name777".isalpha()
```

```
False
```

```
"name777".isalnum()
```

```
True
```

```
string = 'Cloudy ML'
```

```
string.isalpha()
```

False

```
string = 'CloudyML'  
string.isalpha()
```

True

```
str_ = "C++ is easy to learn"  
str_
```

```
'C++ is easy to learn'
```

```
### Replacing substrings ###  
str_.replace('C++', 'Python')
```

```
'Python is easy to learn'
```

```
### using Split function ###  
'Python is easy to learn'.split()
```

```
['Python', 'is', 'easy', 'to', 'learn']
```

```
'Python is easy to learn'.split('easy')
```

```
['Python is ', ' to learn']
```

## Functions

```
# Funntion-1  
YouTubeVideo('https://www.youtube.com/watch?v=cjtAbp-Ux-w&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```

### 15 Function



# Funntion-2

YouTubeVideo('https://www.youtube.com/watch?v=cjtAbp-Ux-w&list=PLsR\_0x6BuM-HJZT6Xzj-088N4h

## 16 Function



- Purpose of functions is to group some particular lines of code that needs to be executed multiple times
  - keyword `def` introduces a function definition and it must be followed by function name
- Function is Reusable piece of code
- We create function to solve specific problem

```
"""
def function_name():
    stetement(s)
"""

'\ndef function_name():\n    stetement(s)\n'
```

```
# define a function
def welcome_message(name):
    """
    This is to represent docstring,
    optionsl documentation is a good
    programming practice
    """
```

```
# call a function with Mukesh and input
```

```
Mukesh Welcome to Python Funcions!!
```



```
#observe the output
print(welcome_message.__doc__)
```

```
This is to represent docstring,
options1 documentation is a good
programming practice
```

```
#observe the output
print(print.__doc__)
```

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

- def Keyword marking start of function
- function name to uniquely identify function
  - function naming follows same rules of writing identifiers
- parameters (arguments) to pass values to a function => totally optional
- () paranthesis
- colon (:) start of function
- documentation string (docstring) describe's what function does => totally optional
- return statement returns a value from function => totally optional
- inside colon is function definition it should always be present before function call or get an error

## ▼ Type of Functions

Built-in	User-Defined	Lambda Functions	Recursion Function
print()	add_two_number(defined by user i.e. you)	``	``
range()	``	``	``
append()	``	``	``
extend()	``	``	``

```
### using print ###
# function to add two number which are as 3 and 4
def two_number_addition():
    # in total variable store addition of 3 + 4
    # print total variable
```

```
### using return ###
```

```
def two_number_addition_with_return():
    total = 3 + 4
    return (total) #returning value of total
```

```
# observe output
```

```
print_output = two_number_addition()
```

```
7
```

```
# see this variable cannot store above value as value was not saved
print_output
```

See above variable cannot store above value as return was not used in function, so every calculation is being done inside the function `two_number_addition()` and if you come out of function those values do not exist, to solve this problem we use `return`, after using `return` we can store output of function for other use in our program

```
return_output = two_number_addition_with_return()
```

```
# as return was used in two_number_addition_with_return() function so we can take out value
return_output
```

```
7
```

Here in above code value to function is being returned and we are saving that returned value to a variable `return_output` for maybe any other further use.

## ▼ Passing Parameters to function

```
# defining a function
def welcome_message(name):
    return 'Welcome to Python Functions!! ' + name
```

```
# calling function in python with name Akash
```

```
'Welcome to Python Functions!! Akash'
```

```
# reusing above function (see it as a login window of a website)
welcome_message('Krishna')
```

```
'Welcome to Python Functions!! Krishna'
```

## ▼ *Passing Default Parameter value to function*

Works in case you dont provide any value to the required parameter

- This is good practice to provide default value to a parameter
  - As it reduces many errors

```
def welcome_message(name='Learner'):
    return 'Welcome to Python Funcions!! ' + name
```

```
# calling function in python
welcome_message('Akash')
```

```
'Welcome to Python Funcions!! Akash'
```

```
# not providing parameter value, now it will take default one i.e. Learner
welcome_message()
```

```
'Welcome to Python Funcions!! Learner'
```

## ▼ *Positional Arguments*

Most arguments are identified by their position in function call

- Say `print(x,y)` will give different results from `print(y,x)`

What ever sequence is given while defining a function values must be taken in that sequence only

- Otherwise use argument name (**keyword arguments**) to take values
- We first define positional argument and then keyword arguments

```
#### Observe every output from here onwards ####
```

```
def subtraction_function(small_number,large_number):
    difference = large_number - small_number
    return difference
```

```
# passing arguments in right order
subtraction_function(7,84)
```

```
77
```

```
# passing arguments in wrong order (produces negative result)
subtraction_function(84,7)
```

```
-77
```

```
# always pass arguments using their name(keyword arguments) then order does not matter
subtraction_function(large_number=84,small_number=7)
```

77

```
# defining positional argument and then keyword arguments => valid in python
subtraction_function(7,large_number=84)
```

77

```
# defining keyword arguments and then positional argument => not valid in python
subtraction_function(large_number=84,7)
```

```
File "<ipython-input-262-731395e41054>", line 2
    subtraction_function(large_number=84,7)
                        ^
```

**SyntaxError:** positional argument follows keyword argument

SEARCH STACK OVERFLOW

## ▼ *Variable Length Arguments*

Used when we need more flexibility while defining functions like we don't know in advance fixed number of arguments

- Python allows us to make function calls with variable length arguments
- In argument use an (\*) astrick sign before argument

##### Observe code from here #####

```
# defining a random function using (*args)
def arguments_function(*args): # => used for positional arguments with variable lengths
    for i in args:
        print(i)
```

```
# passing n-number of positional arguments to function as we are using (*args)
arguments_function(1,2,3,4,5)
```

1  
2  
3  
4  
5

- Everything given to function if it is using \*args, arguments will be saved as tuples

#### Observe every output from here onwards ####

```
# defining a random function using (**kwargs)

def keyword_arguments_function(**kwargs): # => used for keyword arguments will variable le
    for key,value in kwargs.items():
        print(key,value)

# passing n-number of keyword arguments to function as we are using (*args)
keyword_arguments_function(a=1,b=2,c=3,d=4,e=5)

a 1
b 2
c 3
d 4
e 5
```

- Everything given to function if it is using \*kwargs, arguments will be saved as dict

▼ ***Scope of Variables*** means that part of program where we can access particular variable

- Local Variable => variables defined inside a function and can be only accessed from inside of that particular function
- Global Variable => variables defined outside a function and can be accessed throughout program

Let's define a global variable, "global\_variable" outside function

- We will return its value using a function "random\_function" and see that we would be able to access its value using that function also

```
#### Observe every output from here onwards #####
# defining a global variable
global_variable = 'variable outside of function'

# defining function
def random_function():
    # accessing variable which is outside of this function
    return global_variable

random_function()

'variable outside of function'
```

See we can access the data of global variable from Inside of the Function

## => Let's see what will happen if we try to change value of global variable from Inside of the Function

```
#### Observe every output from here onwards ####
# defining a global variable
global_variable = 'variable outside of function'

# defining function
def random_function():
    # changing value of global variable from inside of the function
    global_variable = 'changing variable outside of function from inside of function'
    # accessing variable which is outside of this function
    return global_variable

print(random_function())
print(global_variable)

    changing variable outside of function from inside of function
    variable outside of function
```

It does not mean that we can change global variable from inside the function but this is something different story going on here:

- function named as random\_function()'s, global\_variable have limited scope which is only of inside its function, that is why when we print function it returns the local variable value not the changed global variable value
- variable defined locally is taking precedence over variable defined globally as there name are same

The only shortcomming with local variable is it stayes inside the function, the minute you come out of function you dont get the updated value

So in above code we are not able to change global variable locally, but we have made a new variable which is only working inside of the function.

Approach we shoudl follow is given bellow:

## Updating value of global variable from inside of any user\_defined function using Global Keyword

global keyword tells program that there is no need to make a new variable locally but the variable we are using locally is a global variable, which has benn defined

```
#### Observe every output from here onwards ####
```

```
# defining a global variable
global_variable = 'variable outside of function'

# defining function
def random_function():
    # telling function which variables is/are global
    global global_variable
    # changing value of global variable from inside of the function
    global_variable = 'changing variable outside of function from inside of function'
    # accessing variable which is outside of this function
    return global_variable

print(random_function())
print(global_variable)

    changing variable outside of function from inside of function
    changing variable outside of function from inside of function
```

## Lambda Expressions

Lambda function can have any number of arguments but only one expression, which is evaluated and returned

- Lambda Functions are syntactically restricted to a single expression
- More efficient than function when task of function is small
- Do not use Lambda functions when task is complex
- It works best when you have to call a function more of times but execution task is simple

Write a regular function instead of a lambda when function is to be used multiple times

- Key = Lambda must be used once
- Lambdas are supposed to be used for just once and thus there is no reason to assign a lambda to a variable.
- Major reason for avoiding assigning a lambda to a variable is for debugging/maintainability purposes, especially in a production/teamwork environment
- Use of function is good choice when assignment of variable is compulsory
- While debugging if any error occurs it will point out to the function creating error
- In case of lambda function it will throw `lambda error` and there can be many lambda in our code which makes it difficult to debug error

```
# define a function that adds 100 to number that is passed as an argument
```

```
# lambda function named as increaseBy_100 that adds 100 to number that is passed as an arg

# print value for increaseBy_100(1), using f string
# print value for increaseBy_100(22), using f string
```

```
Values of 1: 101
Values of 22: 122
```

## ▼ Lambda function with two arguments

```
# write a lambda function named as subtract_two_number which can take any two number as in
```

```
-1
```

## ▼ *filter()* function takes in a function and a list as arguments

It filters out all elements of a sequence, for which function returns True

```
#### Observe every output from here onwards ####
```

```
# function to check if given number is even
```

```
def divisible_two(x):
    if x%2 == 0:
        return True #odd
    else:
        return False #not odd
```

```
# given list to filter for odd number
check_list = [4,8,0,11,456,777,999,777]
```

```
#### Observe every output from here onwards ####
```

```
odd_number_filtered_list = list(filter(divisible_two,check_list))
odd_number_filtered_list
```

```
[4, 8, 0, 456]
```

```
# writing lambda function for above code (we need not to define function sepratly)
odd_number_filtered_list_using_lambda = list(filter(lambda x:(x%2==0),check_list))
odd_number_filtered_list_using_lambda
```

```
[4, 8, 0, 456]
```



How easy lamda can make our task now understand

## ▼ ***map()* function** takes in a function and a list as argument

- Function is called with a lambda function and a list
- A new list is returned which contains all lambda modified items returned by that function for each item

```
#### Observe every output from here onwards ####
```

```
check_list = [4,8,0,11,456,777,999,777]
```

```
#increase every element of check_list by 4
```

```
mapped_check_list = list(map(lambda x:x*4,check_list ))
```

```
mapped_check_list
```

```
[16, 32, 0, 44, 1824, 3108, 3996, 3108]
```

```
#### Observe every output from here onwards ####
```

```
check_list = ['north','see','king','himalayas','east']
```

```
# capatilize first letter of each word in the original list
```

```
mapped_check_list = list(map(lambda x: x[0].upper()+x[1:],check_list))
```

```
mapped_check_list
```

```
['North', 'See', 'King', 'Himalayas', 'East']
```

## ▼ **reduce()** Function takes in a function and a list as argument

- The function is called with a lambda function and a list
- A new reduced result is returned
- This performs a repetitive operation over pairs of list

Reduce Function will always take two parameters

```
#### Observe every output from here onwards ####
```

```
#importing reduce using functools
```

```
from functools import reduce
```

```
check_list = [4,8,0,11,456,777,999,777]
```

```
#using reduce function to find sum of numbers in check_list
```

```
sum_of_list = reduce((lambda x, y : x + y),check_list)
```

```
sum_of_list
```

```
3032
```

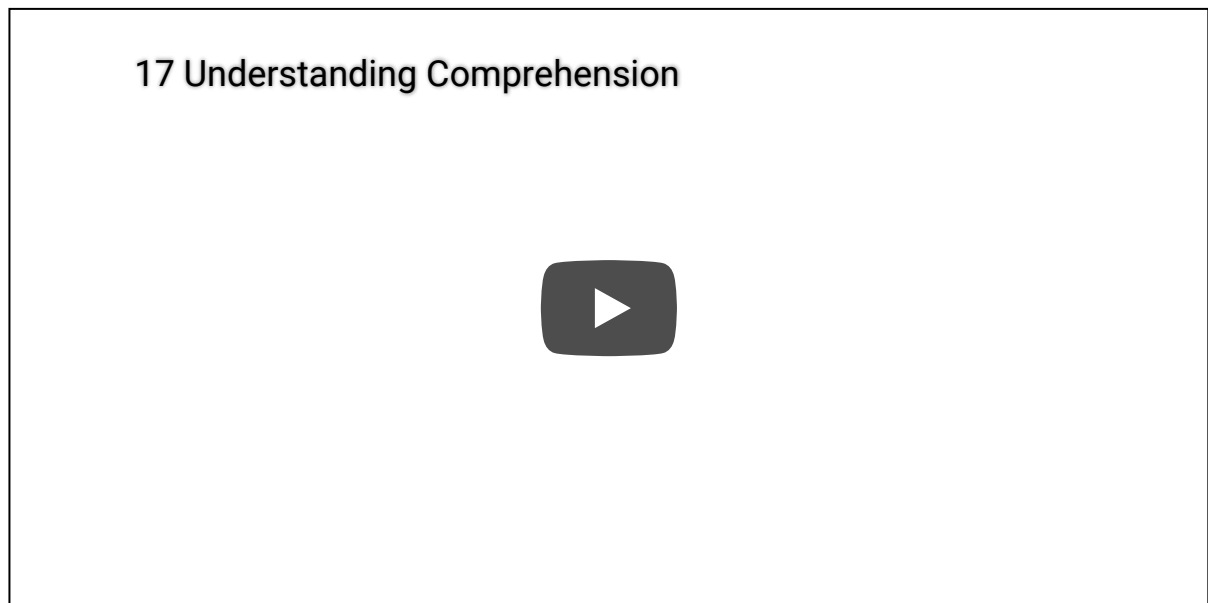
```
#### Observe every output from here onwards ####
```

```
check_list = [4,8,0,11,456,777,999,777]
#using reduce function to find largest number in check_list
largest_number = reduce((lambda x, y: x if (x > y) else y ),check_list)
largest_number

999
```

## Comprehensions

```
# Comprehensions
YouTubeVideo('https://www.youtube.com/watch?v=Cs8wG--s4z4&list=PLsR_0x6BuM-HJZT6Xzj-088N4h
```



**BELOW CODE IS GIVEN FOR YOU TO OBSERVE HOW COMPREHENSIONS WORK.**

### Type of Comprehensions:

1. List Comprehension
2. Set Comprehension
3. Dict Comprehension

#### ▼ 1. List Comprehension

Syntex:

**[expression for item in iterable]**

### Expanded Form

for item in iterable:

....expression

**Converet all element of below tuple in upper case and change datatype into list**

```
animals = ('bird', 'snake', 'dog', 'turtle', 'cat', 'hamster')
type(animals)
```

tuple

# good => way using list comprehension

```
[x.upper()for x in animals]
```

```
['BIRD', 'SNAKE', 'DOG', 'TURTLE', 'CAT', 'HAMSTER']
```

# Not Good way => using list comprehension

```
animalss = []
for x in animals:
    animalss.append(x.upper())
animalss
```

```
['BIRD', 'SNAKE', 'DOG', 'TURTLE', 'CAT', 'HAMSTER']
```

## List Comprehension for Conditional statement or For Filtering

Syntex:

**[expression for item in iterable if some\_condition]**

Expended Form:

for item in iterable:

**\_ if some\_condition:**

**\_\_ expression**

#find the prime number and square them

```
primes = [2,3,4,5,6,8,7,9]
```

# good

```
[x*x for x in primes if x%2 == 0]
```

```
[4, 16, 36, 64]
```

```
# not good
ls = []
for x in primes:
    if x % 2 == 0:
        ls.append(x*x)
ls

[4, 16, 36, 64]
```

**For more complex conditions we can use function along with**

```
#find from below given animal which have 4 leags

animals = ('bird','snake','dog','turtle','cat','hamster')

# good
def has_4_legs(animal):
    return animal in ('dog','turtle','cat')

[x for x in animals if has_4_legs(x)]

['dog', 'turtle', 'cat']
```

```
# not good
l = []
for x in animals:
    if has_4_legs(x):
        l.append(x)
l

['dog', 'turtle', 'cat']
```

## ▼ 2. Set Comprehension

- Set have unique element
- Only use curly bracket then that of square bracket

Syntax:

**{expression for item in iterable}**

```
# find odds
numbers_set = (1, 34, 5, 8, 10, 12, 3, 90, 70, 70, 90)

unique_even_numbers = {number for number in numbers_set if number%2 == 0}
unique_even_numbers

{8, 10, 12, 34, 70, 90}
```

## ▼ 3. Dict Comprehension

Syntax:

**{key\_expression : value\_expression for item in iterable}**

```
words = ('python', 'is', 'a', 'big', 'snake')
```

```
len_words = {word : len(word) for word in words}
len_words
```

```
{'a': 1, 'big': 3, 'is': 2, 'python': 6, 'snake': 5}
```

```
# accessing key which start with p letter
```

```
len_words_p = {word : len(word) for word in words if word.startswith('p')}
len_words_p
```

```
{'python': 6}
```

## Enumerate

Used for efficient looping

- Use enumerate function when needed an index while looping
- Often used in list comprehensions or for loops and does not spit out tuples itself
- Enumerate is an iterator, this means nothing gets stored
- Processes only one item at a time
- This can be useful when you have large datasets/files.

Syntax:

**enumerate(iterable, start)**

**\_\_\_\_start = starting index**

- Returns a tuple of type (index, values)

Whenever you see within your code `range(len(...))` there is a better way to loop than through using an index

- This is where built-in function `Enumerate` can help

## ▼ Enumerate Working

- Enumerate built-in function assigns an index to every item in an iterable object
- This could be a set, list, string anything that can be iterated over

```
colors = ['red','green','blue','purple']
print(enumerate(colors))
```

```
<enumerate object at 0x7fb8769e7320>
```

```
#enumerate gives us index which starts from 0
for index,start in enumerate(colors):
    print(index,start)
```

```
0 red
1 green
2 blue
3 purple
```

```
# index starting from 1 maybe
for index,start in enumerate(colors,start=1):
    print(index,start)
```

```
1 red
2 green
3 blue
4 purple
```

In above code we define variables index and item and they get assigned to first and second items of each tuple

- This is called sequence unpacking
- We then print index and item of items object

```
# using Enumerate to skip over one index
for index,start in enumerate(colors,start=1):
    if index == 3: # skipping index 3
        continue
    print(index,start)
```

```
1 red
2 green
4 purple
```

## FeedBack

We hope you've enjoyed this course so far. We're committed to help you use "AI for All" course to its full potential, so that you have a great learning experience. And that's why we need your help in form of a feedback here.

Please fill this feedback form

[https://docs.google.com/forms/d/e/1FAIpQLSfjBmH0yJSSA34lhSVx4h2eDMgOAeG4Dk-yHid\\_NMTk3Hq5g/viewform](https://docs.google.com/forms/d/e/1FAIpQLSfjBmH0yJSSA34lhSVx4h2eDMgOAeG4Dk-yHid_NMTk3Hq5g/viewform)

