Commenting Code

= # This is a single line comment

Variables

= variables are containers that store data. phython has no command for declaring variable.

= You use = to assign a value to a variable.

ex

```
age = 21
name = "Mary"

print(age)
print(name)
```

output
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 = Output

+ 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 output
Anil's Age : 22

Shot on AWESOME A05s

```
age, income, savings = 21, 4000, 200
print(" Age", age)
print(" Income", income)
print(" Savings", savings)
```

output
Age: 21
Income 4000
Savings 200

PHYTHON DATA TYPES

Data types
- String
- Numeric
  - Integer
  - Float
  - Complex
- Boolean
- None
- Collection
  - List
  - Tuple
  - Dictionary
  - Set.

- Create Integer & String variables.

```
age = 21
name = "Mary"
print (age)
print (name)
```

output
21
Mary

- Create float & boolean variables

```
# create a float variable
product_price = 3423.45
# create a boolean variable.
success = True.
print (product_price)
print (success)
```

output
3423.45
True

- Checking data types with type() function

```
type(age)        type(name)        type(product_price)
int              str               float
```

Data type conversion

Phython allows to convert one data type to another

Type Conversion

Implicit Conversion — Arithmetic Operation

Explicit Conversion — casting Operation

Implicit conversion

conversion done by python Interpreter with programmers 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)

output:

250.75

<class 'float'>

Note

Explicit conversion is required when you have mixed data types

price = 100 — Numeric data type

tax = "18" — string data type

total_cost = price + tax

output

Type error

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

Shot on AWESOME A05s

Explicit Conversion

String data type Needs to be converted to integer type before adding to another numeric variable

```
price = 100
tax = "18"
total_cost = price + int(tax)
print(total_cost)
print(type(total_cost))
```

→ explicit type conversion of string to integer type

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

Type Casting

- int()
- float()
- str()
- bool()

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

output
10000

~~output~~

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

25.0

```
price = 34.6
str(price)
'34.6'
```
output

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

```
False
True
```
output

What is 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.

Shot on AWESOME A05s

The print() in python

```
print ("Hello world")
Hello world                    output
print ("Hello", "how are you?")
Hello how are you?             output.

fruits=("avocado", "apple", "cherry")
print (fruits)
    ('avocado', 'apple', 'cherry')   output.


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

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

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

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.

print with concatenation

```
print ("welcome" + "To the world of python")
welcome To the world of python   = output
```

Note: You cannot concatenate string & number

can only concatenate str (not "int") to str

Shot on AWESOME A05s

- Concatenating with type casting

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

Your age is 35 = output

{ explicit conversion of no. to string with str() function }

PHYTHON OPERATORS

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Special Operators
- Bitwise operators
- Assignment operators

Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Square

Note

Addition

- Adding 2 integer variables

```
price = 100
tax = 25
total_cost = price + tax
print (total_cost)
```

125 = output

- Adding 1 integer variable and 1 float variable.

```
price = 100
tax = 12.80
total_cost = price + tax
print (total_cost)
```

112.8 = output

Shot on AWESOME A05s

Subtraction

- subtracting two integer variables.

```
price = 120
discount = 35
net_price = price - discount
print (net_price)
```

output
85

- subtracting one integer variable and one float variable.

```
price = 125
discount = 35.5
net_price = price - discount
print (net_price)
```

output
89.5

Note: when two strings are added, the operator basically concatenates the two strings. subtracting two strings does not make any sense.

Multiplication

- Multiplication of 2 integer variables

```
price
quantity = 15
total_cost = price + quantity
print (total_cost)
```

output
1800

- Multiplication of 1 integer & 1 float variable

```
price = 120.50
quantity = 15
total_cost = price + quantity
print (total_cost)
```

output
1807.5

- Multiplication of string with integer

```
value = "India"
count = 3
result = value * count
print (result)
```

IndiaIndiaIndia = output

Shot on AWESOME A05s

Note: Multiplication of 2 strings not possible

Division

- Division using 2 integer variables

```
total_cost = 2000
quantity = 100
price_per_unit = total_cost / quantity
print (price_per_unit)
```

output
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)
```

output
15.6075

Note: cannot divide a string by a number

```
service = "airlines"
values = 3
result = service / value
print (result)
```

Note

error ✓

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)
```

output
565

```
total_cost = 13000.50
quantity = 23
price_per_unit = total_cost // quantity
print (price_per_unit)
```

output
565.0

Shot on AWESOME A05s

using modulus operator to find the remainder

Get remainder

```
total_cost = 2700
quantity = 23
price_per_unit = total_cost % quantity
print(price_per_unit)
```

output
9

using modulus operator to find the remainder

```
total_cost = 2700.50
quantity = 23
price_per_unit = total_cost % quantity
print( price_per_unit)
```

output.
9.5

Get square & power output

Squaring variable

```
width_of_square = 20
area = width_of_square * width_of_square
print(area)
```

output
400

power output

```
value = 2
n = 3
```

value raised to n

```
result = value **n
print(result)
```

8 = output

Runtime Variable

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

output

Enter yourname: steve
welcome steve

```
type(input())
str
```

Shot on AWESOME A05s

Logical operators

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

ex

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

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

```
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 the 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

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

Shot on AWESOME A05s

Data Slicing

"[0:1:2]" = Basic Syntax
start stop step

my str[:4] starting से last tak जाएगा

my str[0:] 0 से last tak जाएगा

mystr[: :step] start से last tak चला जाएगा n step k aur stepping kr dega

a = 'great learning'
a[0]
'g'

: → is used to specify the sequence to be sliced.

a = [:]
'great learning'

a = [: len(a)] → len gives length of the string.
'great learning'

a[0:3]
'gre'

a[2:]
'eat learning'

a[2:11]
'eat learn'

a[1:-1]
'reat learnin'

a[: :-1] → reverse the string
'g ninrael taerg'

revers the string & displace the third last letter
a[: :-1][-3]
'e'

Shot on AWESOME A05s

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'~~ 'GREAT LEARNING'

# convert the string in lower case
a.lower()
'great learning'

# convert the first letter of each string to uppercase
a.title()
'Great learning'

# concatenate each element of s1 with sep
s1 = "08", "03", "2000"
sep = '-'
sep.join (s1) → Returns a concatenated string where each character of s1 separated with sep string which is '-'
'08-03-2000'
↳ output

# split the string, text, using comma as a separator:
text = 'cereals grocery cosmetics'
text.split (' ')
['cereals', 'grocery', 'cosmetics']
↳ output

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

Shot on AWESOME A05s

"*" character

text = " ** Hello ** world ** "

text.strip("*") → Returns a copy of the st with both leading and tr characters removed (based on the string argument passed)

' Hello** world ' = output

print the string by removing leading "*" character

text = "** Hello world"

text.strip("*")

" Hello world " = output

Print the string by removing trailing "*" character

text = "Hello world**"

text.strip("*")

' Hello world ' = output

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
  - Tuples
  - List
  - set

List

A list is a container object

Heterogeneous sequence of elements

Can have duplicates

Mutable

Elements are separated by comma, enclosed in [ ] parenthesis

Shot on AWESOME A05s

Creating lists:

```
# Creating customer empty list
customer = []
print(type(customer))

# list of integers
customer_age = [19, 22, 32]
print(type(customer_age))

# list with mixed datatypes
customer_details = [19, "John", 86.2]
print(type(customer_details))
<class 'list'>
<class 'list'>
<class 'list'>
```

→ output

```
# Creating nested lists
# A list can even have another list as an item. This is called nested list.
# nested list
amountspent_by_customer = ["kellie", [18, 400, 60], ['Items']]
print(type(customers_amount_spent))
<class 'list'> = output
```

```
# concatenate and append in lists.
# concatenating lists
states_in_india = ["Tamil Nadu", "karnataka", "Haryana"]
states_in_usa = ["New Jersey", "Texas", "Alaska"]
print(states_in_india + states_in_usa)
```

output.

['Tamil Nadu', 'karnataka', 'Haryana', 'New Jersey', 'Texas', 'Alaska

```
# append
states_in_usa.append("Florida")
print(states_in_usa)
['New Jersey', 'Texas', 'Alaska', 'Florida'] = output
```

Shot on AWESOME A05s

[More List Operations]

How many elements are there in the list

```
Product_Prices = [120, 450.75, 780.90, 670]
len (Product_Prices)
```

4 = output

what is the maximum value in the list

```
max ( Product_Prices)
```

780.9 = output

what is the minimum value in the list

```
min (Product_Prices)
```

120 = output

Note

```
Product_Prices = [120, '450.75', 780.90, 670, 'Morris']
max (Product_Prices)
```

The max() & min() cannot be used on a list with mixed data types.

Append a new element to a list

```
Retail_Products = ['Beverages', 'cereals', 'Detergents']
Print(Retail_Products)
Retail_Products.append ('Cosmetics')
Print (Retail_Products)
```

['Beverages', 'cereals', 'Detergents'] = output

['Beverages', 'cereals', 'Detergents', 'cosmetics'] = output

The new element gets inserted at the end of the list

Remove an element by a specified index pos

Using pop()

```
Retail_Products = ['Detergents', 'Beverages', 'Cereals',
                   'Cosmetics']
Retail_Products.pop(1)
Retail_Products
['Detergents', 'Cereals', 'Cosmetics'] = output
```

Using remove()

```
Retail_Products = ['Detergents', 'Beverages', 'Cereals',
                   'Cosmetics']
Retail_Products.remove('Beverages')
Retail_Products
['Detergents', 'Cereals', 'Cosmetics'] = output.
```

** it removes the first occurrence of the element in case the element occurs multiple times.

Reverse a list

```
Retail_Products = ['Detergents', 'Beverages', 'Cereals', 'Cosmet
Retail_Products.reverse()
Retail_Products
['Cosmetics', 'Cereals', 'Beverages', 'Detergents'] = out
```

```
Average_Prices = [100, 245.50, 675, 121.75]
Average_Prices.reverse()
Average_Prices
[121.75, 675, 245.5, 100] = output
```

Shot on AWESOME A05s

Note: The reverse() and the parameter reverse in the sort() are two different operations.

```
a = [10, 12, 11, 16, 71, 12, 9, 56, 18]
print("The original list is:", a)
#sort
a.sort(reverse=True)
print("The sorted list is:", a)
```

The original list is: [10, 12, 11, 16, 71, 12, 9, 56, 18]
The sorted list is: [71, 56, 18, 16, 12, 12, 11, 10, 9]

```
a = [10, 12, 11, 16, 71, 12, 9, 56, 18]
a.reverse()
print("The reversed list is:", a)
```

The original list is: [10, 12, 11, 16, 71, 12, 9, 56, 18]
The reverse list is: [18, 56, 9, 12, 71, 16, 11, 12, 10]

List [Start, Stop, Step]

Start Position — End Position — increment

red	green	blue	yellow	white	black
0	1	2	3	4	5

red	green	blue	yellow	white	black
-6	-5	-4	-3	-2	-1

Product_Prices = [120, 450.75, 780.90, 670, 189]
Product_Prices [0 : 3]
[120, 450.75, 780.9] = output

Product_Prices = [120, 450.75, 780.90, 670, 189]
Product_Prices [3 : 4]
[670] = output

Shot on AWESOME A05s

TUPLE

- Another type of container object
- Heterogeneous sequence of elements
- Can have duplicates
- Immutable
- Elements are separated by comma, enclosed in () parethesis

Defining a tuple with numeric values

```
Item_Prices = (120, 450.75, 890, 950)
Item_Prices
(120, 450.75, 890, 950) = output
```

Defining a tuple with string values

```
OS = ("windows", "Linux", "MacOS", "Unix")
OS
('windows', 'Linux', 'MacOS', 'Unix')
```

Defining a tuple with mixed data types

```
Product_n_Prices = ('Tetley', 120, 'Nescafe', 450.75, 'Ariel'
                    780.90, 'Twinnings', 670)
Product_n_Prices
('Tetley', 120, 'Nescafe', 450.75, 'Ariel', 780.9, 'Twinnings',
                                                    670)
```

Sort a tuple in ascending order

```
Product_Prices = (120, 450.75, 780.90, 670)
Sorted (Product_Prices)
[120, 450.75, 670, 780.9]
```

Sort a tuple in descending order

```
Product_Prices = (120, 450.75, 780.90, 670)
sorted(Product_Prices, reverse=True)
[780.9, 670, 450.75, 120]
```

Shot on AWESOME A05s

List has a sort() function but a tuple does not have a sort() function. However, you can use sorted() function to sort both a list & a tuple

Note: The sorted() returns a list even if you pass a tuple. Sorted() function returns a list object

```
Product_Prices = (120, 450.75, 780.90, 670)
x = sorted(Product_Prices, reverse = True)
type(x)
Print(x)
[780.9, 670, 450.75, 120]
```

store the result of the sort function in x and check the type of the x variable.

Repeat a tuple

```
tuple_numbers = (10,25)
N = 3
tuple_repeat = tuple_numbers * N
tuple_repeat
(10, 25, 10, 25, 10, 25)
```

Concatenate tuple

```
tuple_heights = (151, 163, 162, 176, 158)
tuple_weights = (76, 65, 74, 56, 85)
new_tuple = tuple_heights + tuple_weights
new_tuple
(151, 163, 162, 176, 158, 76, 65, 74, 56, 85)
```

Shot on AWESOME A05s

Concatenating 2 tuples with numeric values only

```
Prices = (100, 200, 350, 95)
Taxes = (10, 20, 35, 9.50)
Combined_Tuple = Prices + Taxes
Combined_Tuple
(100, 200, 350, 95, 10, 20, 35, 9.5)
```

Concatenating 2 tuples. One tuple with numeric values only and the other with string values only.

```
OS = ("Windows", "Linux", "MacOS", "Unix")
Version = (1.0, 2.0, 1.5, 2.0)
Combined_Tuple = OS + Version
Combined_Tuple
('Windows', 'Linux', 'MacOS', 'Unix', 1.0, 2.0, 1.5, 2.0)
```

Concatenate tuple with mixed data types

```
Tuple_with_mixed_Types = (100, "Windows", "Linux", "MacOS",
                          650, "Unix", 550)
Version = (1.0, 2.0, 1.5, 2.0)
Combined_Tuple = Tuple_with_mixed_Types + Version
Combined_Tuple
(100, 'Windows', 'Linux', 'MacOS', 650, 'Unix', 550, 1.0,
 2.0, 1.5, 2.0)
```

Tuple with membership operator

```
tuple_numbers = (10, 25, 35, 45)
print(25 in tuple_numbers)
True
```

Shot on AWESOME A05s

Note: Because tuple is immutable, a tuple needs lesser memory than lists.

Lesser memory fragmentation is better in terms of performance. Python can reuse a tuple which results in reduction of memory fragmentation and can speed up memory allocations. If a tuple is no longer required and has less than 20 elements, instead of deleting it permanently, python reserves it temporarily

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, item.

```
Dictionary_list = {'name' : 'Ariel',
                   'hobbies': ['painting', 'singing', 'cooki
```

key, value

```
# Create a new dictionary object
books = {
        "R" : 480,
        "Phython" : 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

Nested Dictionary

```
# Creating a nested dictionary object
books={'python':{'chapter_1': 'Introduction to Python'}}
```

```
# Accessing elements from nested dictionary
books['python']['chapter_1']
```

'Introduction to Python'

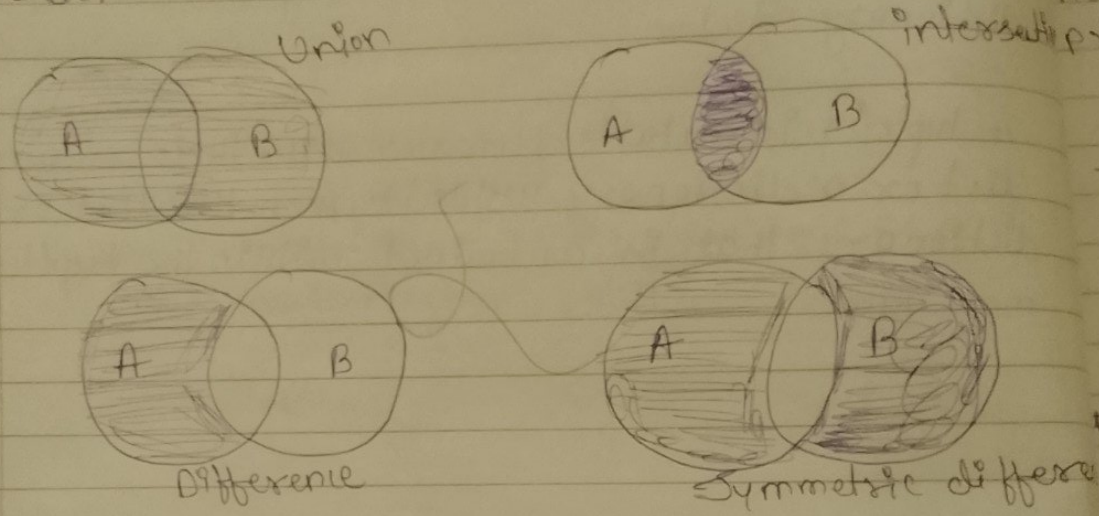
Books { No { A, C, R
Ne { A, C, R
N2 { A, C, R

Books['No']['A']

Books['No']

Sets

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




```
# Create a set
a = {2, 3, "hello"}
print(a)
print(type(a))
```

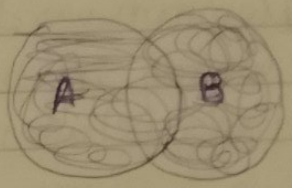
{2, 3, 'hello'} = output
<class set>

```
# initialise the 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 a set of words

sen_1_words = set(sentence_1.split())
sen_2_words = set(sentence_2.split())
```

```
# number of unique words in sentence_1
no_of_unique_words_in_sentence_1 = len(sen_1_words)
print("The number of unique words in 1st sentence:", no_of_
        unique_words_in_sentence_1)

# number of unique words in sentence_2
no_of_unique_words_in_sentence_2 = len(sen_2_words)
print("The number of unique words in 2nd sentence:",
        no_of_unique_words_in_sentence_2)
```

The number of unique words in 1st sentence: 13
The number of unique words in 2nd sentence: 8

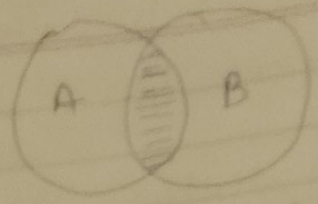
Union




```
# retrieve the unique words in
  both sentences
Unique_words = sen_1_words.union(sen_2_words)
# number of unique words

print("No. of uniquewords in both the sentences:", len(
        unique_words))
print('Unique words in both the sentences:', unique_words)
```

output
No. of unique words in both the sentences: 19
Unique words in both the sentences: {'is', 'in', 'know', 'except', 'the', 'day', 'strength', 'thoughts', 'not', 'comes', 'history', 'you', 'nothing', 'with', 'There', 'new', 'and', 'do', 'word'}

Intersection



# print the common words in both the sentences
common_words = sen_1_words.intersection(sen_2_w
# number of unique words
print ('No. of common words in both the sentences:',
len(common_words))
print ('The common words between the two senten
', common_words)

No. of common words in both the sentences: 2
The common words between the two sentences: {'now', '

difference

print all the words in the sentence_1 that are not in sentence_2

sen_1_diff_sen_2 = sen_1_words.difference(sen_2_words)
print ('words in sentence_1 that are not in sentence_2:',
sen_1_diff_sen_2)

words in sentence-1 that ~~are~~ not in sentence_2: {'is', 'd
'There', 'except', 'history', 'not', 'know', 'wo
'you', 'nothing'}

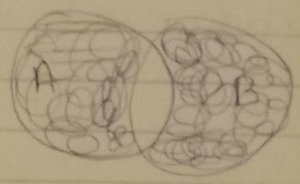
print all the words in the sentence_2 that are no
sentence_1

sen_2_diff_sen_1 = sen_2_words.difference(sen_1_wo
print ('words in sentence_2 that are not in sentence_1:',
sen_2_diff_sen_1)

Shot on AWESOME A05s

output = words in sentence_2 that are not in sentence_1:
{'with', 'day', 'thoughts', 'strength', 'and', 'comes'}

Symmetric difference



symmetric_diff = sen_1_words.symmetric_difference(sen_2_words)
print(symmetric_diff)

{'is', 'with', 'in', 'know', 'There', 'except', 'history', 'day', 'strength', 'thoughts', 'and', 'not', 'comes', 'do', 'world', 'you', 'nothing'}

Conditional Statements

- if statement ←
- if else statement ←
- if elif else statement ←
- Nested if else ←

Syntax

```
if condition:
    statement 1
    statement 2
    .....
```

if statement

```
if (4**2 >= 16):
    print("It is true")
```

It is true! = output

```
if (4**2 < 100):
    print("This won't run.")
```

This won't run = output

For loop

read a string and print the elements one by one

```
nm = 'India'
for i in nm:
    print(i)
```

output

```
I
n
d
i
a
```

While loop

get the list and print input one by one

```
cars = ['Maybach', 'Audi', 'BMW']
i=0
while i<len(cars):
    print(cars[i])
    i+=1
```

Maybach
Audi
BMW
} output

Shot on AWESOME A05s

List Comprehension

- List comprehension provides a concise way to cr
  another list
- Using list comprehensions reduces the numb
  code lines making it. It look more elegant
- It is possible to use conditions in a list comprehe
- The list comprehension always returns a lis

It consists of [ ] containing

an input sequence
a variable representing numbers of the ~~output~~ input sequence
a conditional expression (optional)
an output expression producing a list

variable — filter condition

syntax [output for i in list if condition]
Example [i**3 for i in [1, 2, 3, 4] if i>2]

output expression — input sequence expression.

List comprehension v/s loops

```
num_square = []
for i in range(5):
    num_square.append(i*i)
num_square
[0, 1, 4, 9, 16] = output
```

→ create an empty and append value one by one using for loop.

```
num_squares = [i*i for i in range(5)]
num_squares
[0, 1, 4, 9, 16]
```

→ list compr

Shot on AWESOME A05s

Find strings in a list that are longer than 9 character using list comprehension

Products = ['Apparels', 'Groceries', 'Cosmetics', 'Beverages', 'Detergents']

Product_Category = [p for p in Products if len(p) > 9]

Product_Category

['Groceries', 'Detergents'] = output

Note: A list comprehension helps reduce three lines of code into single line of code

Python allocates the list's memory before appending the elements to the list. This helps avoid resizing the memory at runtime. This makes a list comprehension faster.

Built in functions

Built in functions are those that are already defined in the python library

User-defined functions

A function that a user defines in a program is known as user-defined function

A user can give any name to a user defined function. However a function name should not have space or special character

Note: You cannot use the python key words as function name.

Shot on AWESOME A05s

The def keyword

In python, a user-defined function is created using the def keyword followed by the function name.

Defining user defined function

Step① Declare the function with the python keyword def followed by a function name desired by the user

Step② write the arguments inside the parentheses of the function

Step③ End the declaration with a colon

Step④ Add the program statements that need to be executed

Step⑤ End the function. Note a function can be ended with or without a return statement

keyword def — functionname

```
def area-of-triangle(base, height):
    return (0.5 * base * height)
```

value to be returned — input parameters.

write your first function using the def keyword

```
#Create a function 'greeting' to display greetings for the day

def greetings():
    print("Hello, have a wonderful day ahead!")
# call the function
greetings()
```

Hello, have a wonderful day ahead! = output.

Shot on AWESOME A05s

Functions Arguments

Types of function arguments
- Required Arguments
- Keyword Arguments
- Default Arguments
- Variable-length Arguments
- Variable length keyworded Arguments.

Required Arguments

(i) in case, if the function does not require the argument and is passed it will throw an error

```
#create a function 'greeting' to display greetings for the day
def greetings():
    print("Hello, have a wonderful day ahead!")
# call the function
greetings()
Hello, have a wonderful day ahead!
greetings()    # call the function
Hello, have a wonderful day ahead!
greeting(name)
Name error
: name 'greeting' is not defined.
```

(ii) In case, if the function requires the argument and is not passed it will throw an error

```
# create a function 'hello' to display greetings for the day
# the function has argument 'name'
def hello(name):
    print("Hello," + name + "have a wonderful day ahead!")
hello()
Type error
: hello() missing required positional argument: 'name'
```

Shot on AWESOME A05s

Lambda function

- Lambda functions are anonymous (i.e to say they have no names
- The lambda is a keyword
- It is a simple one line function
- No def or return keyword to be used with a lambda function

A lambda can have multiple arguments separated by commas

lambda arguments : expression

Every lambda begins with the "lambda" keyword

A colon precedes the expression

expression always returns an object

Syntax (1) [value for value in iterable]

Ex- i for i in range (10,16)

Note: Using lambda function to reduce code size

Normal python code

```
def fun(x,y):
    if(x>y):
        return x
    else:
        return y
print(fun(3,4))
```

4 = output

Using lambda function

```
#The general method shown on the # left can also be rewritten # using lambda
fun = lambda x,y : x if x>y
    else y
print(fun(3,4))
```

4 = output

Wrong Usage of lambda

You will need to declare the variable which you may use inside the lambda function. In the below example, we use variable, b, without declaring it.

```
#multiplication - wrong usage of variable b
x = lambda a,c : a*b
print(x(5,4))
```

NameError: name 'b' is not defined.

Shot on AWESOME A05s

Variable length keyworded argument

*wargs

```
def average(*nums):
    add = sum(nums)
    count = len(nums)
    return add/count
average(33,45,12,56,89,32,65,11)
        41.65
average(45,32,77)
        51.3333336.
```

Q define a function which takes any number of arguments and returns the mean of the largest and smallest argument.

```
def min_max(*ages):
    maxi = max(ages)
    mini = min(ages)
    return(maxi+mini)/2.
min_max(32,45,91,56,32)
43.5 = output
```

Variable length keyworded arguments using ** kwargs

```
def description(**info):
    return info
description(Name='Anjali', Gender='Female',
            Salary = 1600000)
def description(**info)
    for i in info.keys():
        print(i)
    for i in info.values():
        print(i)
description(Name='Muhesh', Designation='Manager',
            Age = 27)
```

output

Name Designation Age Muhesh Manager 27

Shot on AWESOME A05s