Artificial Intelligence

Wednesday, July 29, 2020 12:35 PM

Artificial Intelligence: Replica of human intelligence in machine.
Two types:
Specific AI
Specific to job. Self-driving car.
General AI: Human intelligence is General AI.

Machine Learning: recognize pattern using algorithm.
Deep Learning: more specifying ML.
DATA Science and AI includes both of above

Two approaches:
Going from base or root: mathematical algorithm.
Application: Focusing on apps.

2nd Lecture:

ML? Learning capability of computer.
Learning capability of computer to recognize, compute, to decide and to take remedial measures.

3rd Lecture:
Website: Teachable machine: to teach ML.
Image project,
3 part: Add data (pre-prosessing), training, preview (post-proceessig).

Export model: give u code, can be used in python.

4th Lecture:
Supervised learning(ML playground).

Data required from users to generate pattern using K Nearest Neighbors algorithm(also used in Amazon recommendation system to collect data)

5th Lecture:

Data Science:
Science of data. Performing actions on data.
3 Actions of DS:

1. **Pre-processing**

Have to prepare data for the actions to be performed. It itself is field: Data Engineering.

1. **Action**: Algorithm.
2. **Post** Processing: presentation of data.

Machine Learning:
Learn from the data to predict (LP)
Extension of Action part of DS.
Labeled data
Unlabeled data.

Types of ML: 3 types.

1. **Supervised Learning**

Data is given for classification and prediction.
Classification , regression: prediction.

1. **Unsupervised Learning.**

Here unlabeled data is given. Here, task is not given but Machine itself accrue result an666666d collects the similar types of objects. It is also called CLUSTERING.

1. **Reinforced Learning.**

Data is given, task not given but set of rules(E.g. Alpha Rules).

All of it come in the ambit of **Artificial Intelligence** (Maza aya, haaiiin)

6th Lecture:
Summery
Machine Learning: Ability of computer on the basis of data.

Section 3

Artificial Intelligence

Saturday, August 01, 2020 12:29 PM

7th Lecture:
Preprocessing: Data collection
Processing: Modelling, applying of ML model
Post Processing: Deployment of Model.

(ML encircles around Modelling:)

Setting of Rear view mirror and ML:
Problem, Definition:

Information: Required to solve problem.
What type of data or information.
Structure data, unstructured data, static data , time series data.

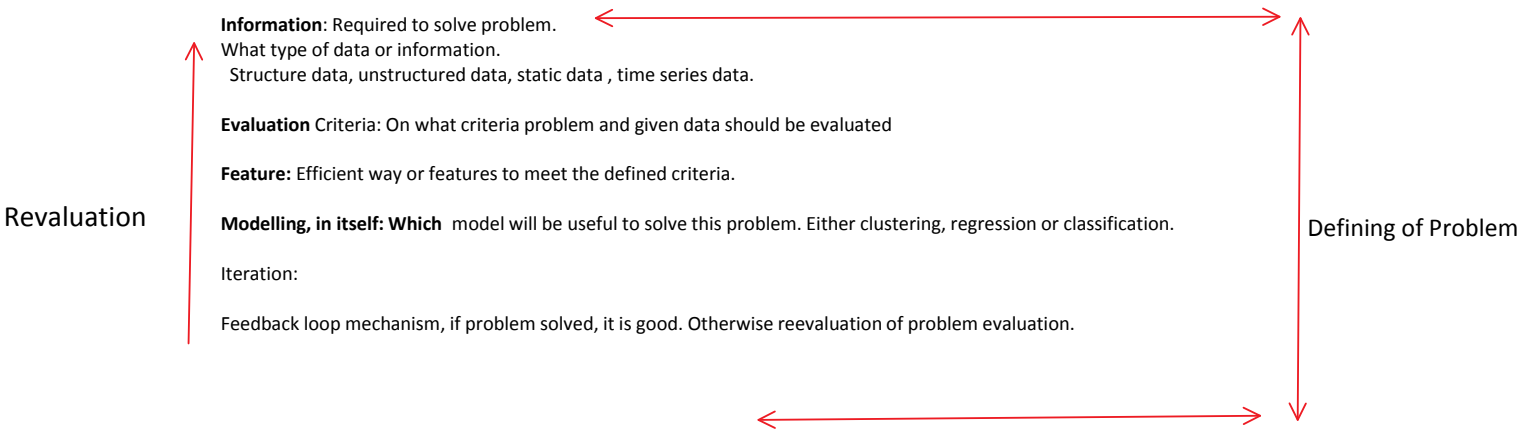
Evaluation Criteria: On what criteria problem and given data should be evaluated

Feature: Efficient way or features to meet the defined criteria.

Modelling, in itself: Which model will be useful to solve this problem. Either clustering, regression or classification.

Iteration:

Feedback loop mechanism, if problem solved, it is good. Otherwise reevaluation of problem evaluation.



LECTURE 8

Types of ML and how they solve problems.

Supervised
- Classification: labelled data
- Regression : prediction on the basis of available data

Unsupervised
it is used when labelled data is not available.

Transfer
Transferring of learning method for different projects of same nature.

Reinforce
Intensify the selection of required subjects OR chose what is required.

Lecture 8

Type of Data:

Structured Data: comma separated values,

Unstructured Data: There is not pattern of rows and columns. E.g Image, audio.

Types of structured and unstructured data
-Static data: fixed values.
-Time Series Data: vary with time eg stock market.

Workflow will be normally from static to time series.

data (open in Jupiter note book) VISUALIZE (panda library, matplot liberary) Application on ML model (Sickit Learning)

LECTURE 10

Problem:
Data:
Evaluate: subject to requirement
Test data set

Training data set
Tuning is done of validation data
Feature
Feature variable :
-numeral
-categorical
Target variable: which is predicted
Derived Feature: extracted from existing data

Modelling
- choose and train
  * training data
- tune and model
  *validation data
- model compare
  * test data.

Iteration

Three steps of modeling.
Data is required for every stage.
Choose and train Model:
Data to train: Train Data
E.g.:Reading during semester is like training.

Tune Model:
Data required for tune: Validation data
E.g.: Practice exam before final exam is validation

Model Comparison:
Data for model: Test Data
E.g.: Final exam is actual test data. Or model comparison.

Overfitting: Accuracy more than the requirement.
Under fitting: Accuracy less than the requirement.

Model Tuning:
Hyper parameters:
Through this performance could be altered to acquire required result.

Validation of data:
How model would be compared?
How would one get to know that model is working accurately?
Model Accuracy tells this. Accuracy on validation of data must be lower than accuracy achieved during training.
MAC>>>TACC : Overfitting
MAC<<<TACC : UNDERFITTING
Both overfitting and under fitting are not required.
We have to define a model which almost perfectly predict outcomes and represent the data effectively.

Iteration/Experimentation
After having fairly good result of validation of model. This model can further be improved by working on its:
-Accuracy
-Computational Cost

Anaconda :
Whole framework would be applied here.
Data analysis, evaluation and feature task would be done in **Panda**, **Mat plot and Numbpy liberaries**.
Modeliling would be carried out in **Tenser flow Pytorch and Ski Learn.**
Jupitor Tool Book

Machine Learning:

1-Problem Definition (constitute **DATA SCIENCE**)

2-Data: to solve problem
3-Evaluation.
4-Feature: generating another column of data using the available columns of data.

These three constitute **Data Analysis**

5-Modeling: mathematical algorithm. (constitute **Machine Learning**)

6-Iteration: Improving accuracy of model through experimentation.

Tools Required for Data Science:

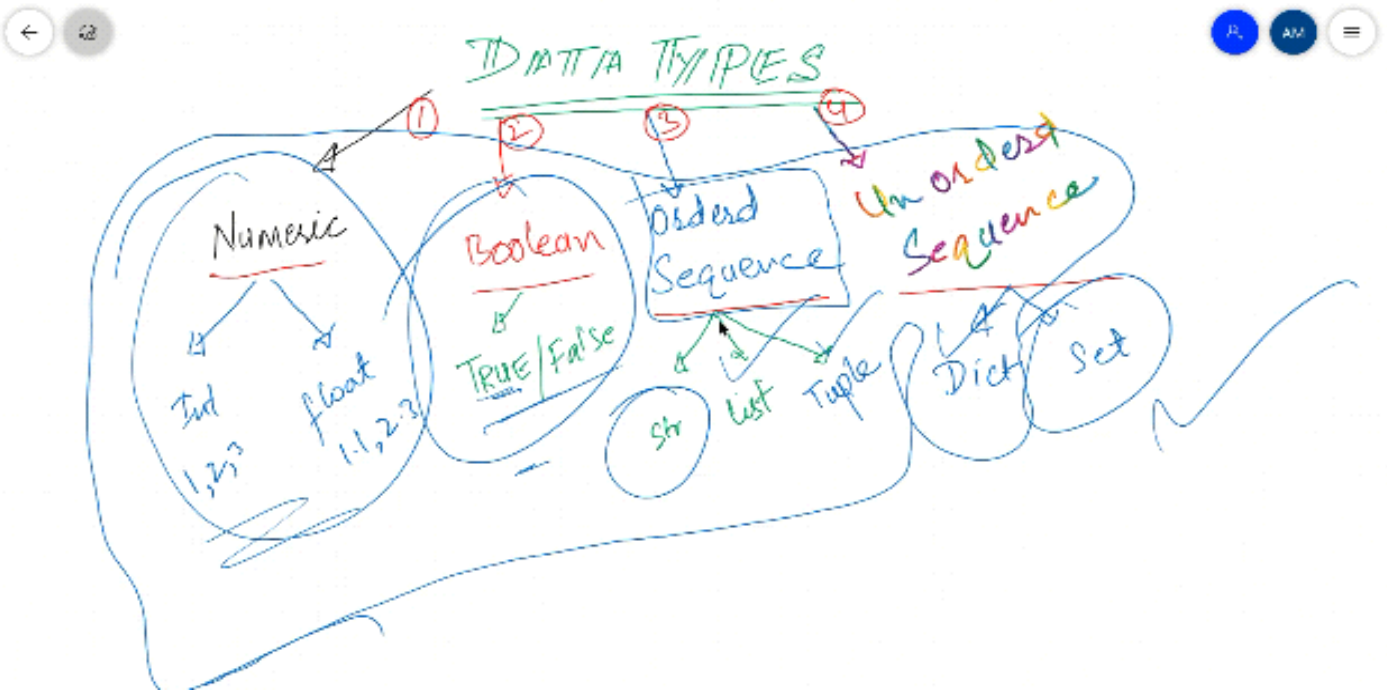
Miniconda: In this software data would be analyzed.

Libraries needed for this
Pandas
Numpy
Matplotlib

For Machine Learning
Tenorflow
SK-Learn
pytorch

Python Language: Data Structures

Wednesday, October 07, 2020 6:37 AM



Extension of python: .py
Std: standard output.

Syntactical difference.

Print(): Built in function.
Helloword: String.

Print('Hello' + 'Awais')

(we can store string or name in memory)

Through this
name = 'Awais'
print('Hello ' + name)

Python 2 is older version and python 3 is new version. There is syntactical difference between these two.

Requirement to write code:
1-Data(types),
2-Action

What are the best practices to write code?

Programing Vocabulary is indispensable.

Data Types
Numeric
int(1,2,3)
float(1.1,2.3)

Bolean
True/False

Ordered Sequence
Str(string),
List,
Tuple.

Unordered Sequence:
Dict(Dictionary),
Set.

None:

Operators:
+ - / * // (to convert floating value to whole number) ** (to take powers) % (remainder)

Arithmetic:
Functions/methods: There are multiple predefined actions these are called methods or functions.
One of the function is

Type().
used to know the type of data.

Check
Print(type(1+2)

Print(type(1/2)

Round()
It coverts floating into round.

Print(round(4.3))

Abs(): to have absolute value not negative value.

Print(abs(-3.5))

Note: Firstly, inner brackets will be solved.

Whole number takes smaller space as compared to floating value.

Best way to learn programming:

80/20 rule:
20 percent of things covers 80 percent workspace.

This rule is also applied in programming that 20 percent of programming is applied on 80 percent of projects . So, we have to focus upon that 20 percent.

We need to learn how to take help from google and documentation. As there is no need to learn anything. No need to memorize. No need to know everything about everything.

Mathematical Operator and precedence.
() parenthesis
** powers
*/ multiplication, division.
+ - addition, subtraction.

PEMDAS: Multiplication or division is first solved then addition or subtraction. Parenthesis and power solved before multiplication and division.
Print((20+4)*2**2)

Variables:
Python stored data into a memory whenever it has to perform action it. That assigned memory is known as VARIABLE.
Jb b data pe operation krna hta h to python usko memory main save krta h, us memory ko variable krte hn.

Your lunch box in school upon which your name is also mentioned. That name is VARIABLE while lunch box is memory.
Ap k school k lunch box pe jo apka nam likha hwa hta tha wo variable h r lunchbox itelf aik memory h.

(= Assignment operator)

```
Box =  'shoes'
Print(Box)
```

4 big No's for variables.
Don’t start name of variable with number.
4box: invalid.
Box4: valid

No special charactert. @!$%%()
@box:invalid
box@invalid

No space
My box: invalid
My_box: valid

No variable on the name of built-in-function.

Print = 'My paper'
Print(Print)
Here built in function print is replaced by 'My paper'.

Better to write descriptive name of variables.

#STATEMENT(IT HAS EQUAL SIGN)
A , b = 10, 20
Print(A, B)
(**Reading is from right TO LEFT)**

Augmented Assignment Operator:

```
  a = 10
 a = a + 10
Print(a)
```

Or

```
 a = 10
  a += 10
Print(a)
```

(#firstly defined variable such as "a" must be defined)

Data Type

STRING DATA TYPE:

ORDERED SEQUENCE:

String is sequence of character.
E.g: Hello, I am a good guy. If sequence is changed then meaning would not be conveyed. Look at this, " guy good hello am I"

There are three ways to represent STRING

If there is apostrophe is message, use double inverted commas

```
  message = 'This is Awais' message'
Print(message)
 # will not be read after last inverted command.

 message = "This is Awais' message"

 print(message)
```

2ND WAY

```
 quote= " I remember u said "never give up"
Print(quote)
# Simililar error
```

```
quote= ' I remember u said "never give up" '
Print(quote)
```

3rd way

```
  multi-line = ' ' '
 I remember u said "never give up"
 I remember u said "never give up"
 I remember u said "never give up"
' ' '
Print(multi-line)
```

String catenation: connecting two strings.

"+" :

Operator overloading: single operator is used for more than one operators.

```
message= ' Heloooo'
name = 'Sally'

print(message + name)
```

```
message= ' Heloooo'
name = input('please enter your name:')

print(message + name)
```

Similar type will always be concatenated. String will not be concatenated with number.

```
message= ' Heloooo'
name = input('please enter name:')
Print(type(name))

print(message + name)
```

Check result…

```
message= ' Heloooo'


print(message + 5)
```

There will be error in results.

Type Conversion

Types of input can be converted by putting the input in bracket of that very type.

```
message = 5
n = int(input("please enter digit: "))
```

```
print(type(n))

print(message + n)
```

Conversion of input into num type

```
num_1 = 5
num_2 = int(input("please enter digit: "))
print(type(num_1))
print(type(num_2))

print(num_1+num_2)
```

Conversion input into str type

```
num_1 = str(5)
num_2 = str(input("please enter digit: "))

print(type(num_1))
print(type(num_2))

print(num_1+num_2)
```

Output:

```
please enter digit: 5
<class 'int'>
<class 'int'>
55
>
```

Output

```
please enter digit: 5
<class 'str'>
<class 'str'>
55
>
```

String Formatting:

```
name =  'Mohsin'
price = '10'
print(' Heloo' + name + 'your item price is' + price)
```

)

Result

```
HelooMohsinyour item price is10
```

Print Using .Format Method:{} {} .format(v1, v2)

Above code in compact using string formatting

```
name =  'Mohsin'
price = '10'
print( "Heloo {} your item price is {}"  .format(name, price))
```

OUTPUT
Heloo Mohsin your item price is 10
>
In string formatting, values of different variables can be shown at different places by using ".format(variable 1, variable 2) in "print" function using
----
Another way

```
name =  'Mohsin'
price = '10'
print( "Heloo {1} your item price is {0}"  .format(name, price))
```

o/p
Heloo 10 your item price is Mohsin

Alternative of .format()

```
name =  'Mohsin'
price = '10'
print( f"Heloo {name} your item price is {price}")
```

RESULT
Heloo Mohsin your item price is 10

Indexing:
String is ordered sequence of character (OSC)
Meaning:
(First u must know)
Index is a location identifier in Python. It starts from zero.

```
#to access character,index has to be placed
name = 'abcdef'
print(name[4])
```

Result
e
Because
abcdef
012345

SKIPPING

[START:STOP:SKIP]
Without entering value of SKIP, by default compiler would take 1.(SKIP WALA POINT BY DEFAULT 1 HOTA)
Without START reading begins from zero index
Without STOP index will be read till end

```
name = 'abcdef'
print(name[0:5])
```

O/P
abcde

```
name = 'abcdef'
print(name[0:5:1])
```
o/p
abcde
(no change)

```
name = 'abcdef'
print(name[0:6:2])
```

o/p
ace
Here skipping occurs/

Negative Index
```
name = 'abcdef'
print(name[::-1])
```
fedcba

```
name = 'abcdef'
print(name[-1])
```
o/p
f

IMPUTATION:
This concept is valid on string. This asserts that its basics can't be changed.
A new value to a variable can be given but a value of single index can't be changed. This is imputation.

```
numbers = '0123456'
numbers = 'abcdefg'
print(numbers[1])
```
o/p
b
(**a new value is assigned to variable "number' whose index 1 is printed. But alteration only in this very index 1 is not possible. For instance:**

```
numbers = '0123456'
numbers[1] = b
print(numbers[1])
```

```
Traceback (most recent call last):
  File "<string>", line 4, in <module>
NameError: name 'b' is not defined
>
```
(error occurred)

Methods & Functions:
Str, int, print and type are example of function.

Function is an already written code.

Everything in python in object

Calling function over an object is Method.

```
string = 'never give u'
print(string.capitalize())
```
O/P
Never give u

Here Capitalize function is called over an object String

```
string = 'never give u'
print(string.upper())
```
o/p
NEVER GIVE U

```
qoute = 'never give up'
print(qoute.find('up'))
```

o/p
11
Found index using .find function.

```
qoute = 'never give up'
print(qoute.replace('up','me'))
print(qoute)
```

o/p
never give me
never give up

HERE COMPILER didn't violate the imputation.
.replace() function creates a copy of an object quote and replaces it's 'up' with 'me'. Original is still same.

Bolean
ONLY TWO TYPES exist.
True
False
bool() : only considers 0 and empty as false while any other number or string will be considered as true.

```
light_has = True
print(light_has)
print(bool(0))
print(bool(1))
print(bool('a'))
```

o/p

True

False
True
True

Exercise
Get input from user and encrypt it

Solved by me with extra print() function to show original name after encrypting it.

```
print(' kindly write your name: ')
name = input()
print('your name is '+ name.replace(name, 'hello'))
print (f"original name is {name} but don't tell anybody")
```

O/P

```
kindly write your name:
awais
your name is hello
original name is awais but don't tell anybody
```

Solved by sir

```
name = input('please enter your name ')[::-1]
print (f"hacker is reading {name}")
print (f"hacker is reading {name[::-1]} ")
```

Result
```
please enter your name awais
hacker is reading siawa
hacker is reading awais
```

Data Structure: Used to hold data or container and to hold different objects of python.
Cupboard is hypothetical example of data structure.

List :
[] : square brackets are used to represent list.
List: is itself is data structure. Its indexing is possible which means its items can be changed particularly.
NOTE: A list is a data structure therefore it can store data of all types.
It's an **ordered sequence of object** just like a **string which was an ordered sequence of character.**

```
list_1 = [1,2,3]
list_2 = ['a', 'b','c']
list_3 = [True, False]
print(list_1)
print(list_2)
print(list_3)
```
o/p
```
list_1 = [1,2,3]
list_2 = ['a', 'b','c']
```

```
list_3 = [True, False]
print(list_1)
print(list_2)
print(list_3)
```

CHECK OUT THE TYPE

```
wao_1 = [1,2,3]
wao_2 = ['a', 'b','c']
wao_3 = [True, False]
print(type(wao_1))
print(type(wao_2))
print(type(wao_3))
```

O/P

<class 'list'>
<class 'list'>
<class 'list'>

Approaching the objects:

```
wao_1 = [1,2,3]
wao_2 = ['a', 'b','c']
wao_3 = [True, False]
print(wao_1[0])
print(wao_2[1])
print(wao_3[1])
```

o/p

1
b
False

List within a list:

List within a list is an object and can not only be accessed the whole list but also a particular object of that very list stored at unique index.

```
wao_2 = ['a','b','c',[1,2,3,5],[True, False]]
print(wao_2[4][0])
```

o/p

True

List is ordered sequence of object. Its slicing can also be performed.

```
wao_2 = ['a','b','c',[1,2,3,5],[True, False]]
print(wao_2[0:4])
```

o/p

['a', 'b', 'c', [1, 2, 3, 5]]

Note: here at index #3 whole series is stored as single object.

Slicing/skipping of series within a series is also possible as

```
wao_2 = ['a','b','c',[1,2,3,5],[True, False]]
print(wao_2[0::4])
```

o/p
['a', [True, False]]

Mutability Character of List

```
list_items = ['shoes', 'book', 10, 'laptop' ]
list_items[2] = 100
print(list_items)
```

o/p
['shoes', 'book', 100, 'laptop']

[:] is used to make a copy of original

```
list_items = ['shoes', 'book', 10, 'laptop' ]
new_list = list_items
new_list [3] = 'tablet'
print(list_items)
print(new_list)
```

o/p
['shoes', 'book', 10, 'tablet']
['shoes', 'book', 10, 'tablet']

Note: Although new list is created, but while changing the object of new list a change is also occurred in list_items. For this, [:] is used to make a copy of original list in order to forbade the the original list from being copied.

```
list_items = ['shoes', 'book', 10, 'laptop' ]
new_list = list_items[:]
new_list [3] = 'tablet'
print(list_items)
print(new_list)
```

o/p

['shoes', 'book', 10, 'laptop']
['shoes', 'book', 10, 'tablet']
#check the difference in output. Here list_item is not being changed.

Matrix :
It's defined by creating lists within a list.

```
matrix = [[1],[2],[3]]
print(matrix)
```

o/p
[[1], [2], [3]]

Why need of matrix?

Image has 3 values.
720 x 1080: x & y dimensional. 3rd element is RBG value

List Methods:

Function and methods are already and built-in written code.

Different Methods can be called on a list by pressing/mentioning "." (dot) after it. Suitable methods can be selected from the lists of methods shown by ide.

https://www.w3schools.com/python/python_ref_list.asp

.Insert Method

```
shopping_cart = ['shoes', 'banana', 'orange']
shopping_cart.insert(2,'mango')
print(shopping_cart)
```

O/P
['shoes', 'banana', 'mango', 'orange']

NOTE: **INSERT METHOD DOES NOT RETURN VALUE**

```
shopping_cart = ['shoes', 'banana', 'orange']
new_list = shopping_cart.insert(2,'mango')
print(new_list)
print(shopping_cart)
```

o/p
None
['shoes', 'banana', 'mango', 'orange']

None is showing because inset function is not returning any value to a variable but only creating a change in original list .Returning a value means that particular function is not returning or storing a value upon which it is performing a function.

.APPEND FUNCTION
(append: adding to end of list)

```
shopping_cart = ['shoes', 'banana', 'orange']
shopping_cart.append('u got it')
print(shopping_cart)
```

O/P

['shoes', 'banana', 'orange', 'u got it']

Append doesn't return the value and change occurs in original list.

```
shopping_cart = ['shoes', 'banana', 'orange']
new_value = shopping_cart.append('u got it')
print(shopping_cart)
print(new_value)
```

o/p

['shoes', 'banana', 'orange', 'u got it']
None

.REMOVE METHOD

```
shopping_cart = ['shoes', 'banana', 'orange']
shopping_cart.remove('banana')
print(shopping_cart)
```

o/p

['shoes', 'orange']

Remove Method does not return the value:

```
shopping_cart = ['shoes', 'banana', 'orange']
new_val  = shopping_cart.remove('banana')
print(shopping_cart)
print(new_val)
```

o/p

['shoes', 'orange']
None
Notice: Remove is not returning the value and change occurred in original list.

POP METHOD : it removes the defined value at particular given index. If index is not provided, it will remove the last value of list which is stores at the last index.

```
shopping_cart = ['shoes', 'banana', 'orange']
new_list = shopping_cart.pop()
print(shopping_cart)
```

o/p

['shoes', 'banana']
Last value is popped.

```
shopping_cart = ['shoes', 'banana', 'orange']
new_list = shopping_cart.pop(1)
print(shopping_cart)
print(new_list)
```

o/p

['shoes', 'orange']
Banana

Pop returns the value unlike .remove, . Append, .insert, etc. And creates change in original list.

CLEAR METHOD: TO CLEAR/EMPTY LIST

```
hopping_cart = ['shoes', 'banana', 'orange']
new_list = shopping_cart.clear()
print(shopping_cart)
```

o/p
[]

COUNT METHOD:to know how much time a value appeared.

```
NUMBERS = [1,2,3,11,1,5]
print(NUMBERS.count(1))
```

o/p
2

INDEX METHOD: to know the location

```
NUMBERS = [1,2,3,11,1,5]
print(NUMBERS.index(11))
```

o/p
3

Location finder in defined range .index(find-this, start, stop)

```
NUMBERS = [1,2,3,11,1,5]
print(NUMBERS.index(11,0,4))
```

o/p
3

```
NUMBERS = [1,2,3,11,1,5]
print(NUMBERS.index(11,0,3))
```

o/p
ValueError: 11 is not in list

SORT METHOD: TO ARRANGE NUMBERS IN ORDER

```
NUMBERS = [1,2,3,11,1,5]
NUMBERS1 = NUMBERS.sort()
print(NUMBERS)
print(NUMBERS1)
```

O/P
[1, 1, 2, 3, 5, 11]
None
.SORT DOEST NOT RETURN THE VALUE AND MAKES CHANGES In ORIGINAL LIST THAT'S WHY 'NUMBERS1' IS NOT PRINTING ANYTHING

```
NUMBERS = [1,2,3,11,1,5]
NUMBERS.sort()
print(NUMBERS)
```

O/P
[1, 1, 2, 3, 5, 11]

SORTED() methods: THIS METHOD WORKS AS SORT METHOD DOES BUT IT ALSO RETURNS VALUE

```
NUMBERS = [1,2,3,11,1,5]
NUMBERS0 = NUMBERS.sort()
NUMBERS1 = sorted(NUMBERS)
print(NUMBERS0)
print(NUMBERS1)
```

o/p

None
[1, 1, 2, 3, 5, 11]

.REVERSE METHOD: MIRRORED THE LIST

```
NUMBERS = [1,2,3,11,1,5]
NUMBERS.reverse()
print(NUMBERS
)
```

o/p

```
NUMBERS = [1,2,3,11,1,5]
NUMBERS.reverse()
print(NUMBERS
)
```


So, in all above-stated methods only .POP and .SORTED() returns value.
(Discussed methods which do not return values
are: .insert, .remove, .append, .Clear, .count, .reverse, .sort)

How to create list automatically?

For this, Range function is used.

```
print(range(1,10))
```

o/p
```
print(range(1,10))
```

Created list can be coverted into another type and then printed.

```
print(list(range(1,20)))
```

o/p
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

VALUES IN LIST CAN BE ASSIGNED TO MULTIPLE VARIABLES

```
a,b = [1,2]
print(a)
print(b)
```

O/P
1
2

```
a,b,c = [1,2,3],[4,5],[6]
print(a)
print(b)
print(c)
```

o/p
[1, 2, 3]
[4, 5]
[6]

```
a,b,c,*REMAINING = [1,2,3,4,5,6]
print(a)
print(b)
print(REMAINING)
```

o/p
1
2
[4, 5, 6]

Dictionary
To know value of key…
Its key value pair

{'key1' :1, 'key2':2}

Unlike list and string Dictionary is totally unordered.

```
dictionary ={'key1' :1, 'key2':2}
print(dictionary)
print(dictionary['key1'])
```

o/p

```
dictionary ={'key1' :1, 'key2':2}
print(dictionary)
print(dictionary['key1'])
```

NOTE:
(dictionary[0]) THIS method cant be used to find the value of any variable defined in key.

Index of defined list within dictionary is possible.

```
dictionary ={'key1' :[1,2,3],'key2':True, 'key3': 'wao' }
#to print whole dictionary
print(dictionary)
#to print value at 0 index of list
print(dictionary['key1'][0])
#to print valye of key3
print(dictionary['key3'])
```

o/p

'key1': [1, 2, 3], 'key2': True, 'key3': 'wao'}
1
Wao

```
#QUIZ:LOCATE 'B'
dictionary =[{'key1' :[8,9,5],'key2':True, 'key3': 'wao'
'key4:4' }, {'key1' :[1,2,3],'key2':True, 'key3': 'wao'
'key4:4' },{'key1' :['a','b',3],'key2':True, 'key3': 'wao'
'key4:4' }]

print(dictionary[2]['key1'][1])
#WOOOOOO, SUCCESSFULY SOLVED IT.
```

O/P
b

Dictionary is unchangeable. Key of dictionary is unchangeable or immutable. Generally, list is mutable but when it is added into a dictionary it will be changed by approaching its index.

Which data structure is best?
None is best. All it depends upon the requirement and suitability.

.get Method

```
characteristics = ['name', 'power', 'bonus']
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print('this is rest of the code')
print(dictionary['weapons'])
```

o/p
this is rest of the code
Traceback (most recent call last):
File "main.py", line 8, in <module>

```
   print(dictionary['weapons'])
KeyError: 'weapons'
```
Note: here code is printing "this is rest of code'

```
characteristics = ['name', 'power', 'bonus']
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print(dictionary['weapons'])
print('this is rest of the code')
```

o/p
```
raceback (most recent call last):
  File "main.py", line 7, in <module>
    print(dictionary['weapons'])
KeyError: 'weapons'
```
By changing position of print('this is the rest of the codre'), its output is not showing because compiler is not executing our code lower to the error. IS THERE ANY WAY TO SKIP THIS ERROR AND EXECUTE THE REST OF CODE?

.get method solved this problem.

Look:

```
characteristics = ['name', 'power', 'bonus']
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print(dictionary.get('weapons'))
print('this is rest of the code')
```

o/p
```
None
this is rest of the code
```

.**key Method**
It willl give list of key.
```
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print(dictionary.keys())
```

o/p
```
dict_keys(['player', 'power', 'bonus'])
```

.value method
It will give values of keys

```
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print(dictionary.values())
```

o/p
dict_values(['name', 'full', '2 lives'])

In dictionary: to chechk whether a certain key or value exists in dictionary or not? Result/op will be in bolean.

syntex : 'key in variable.keys()

To check key

```
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print('power' in dictionary.keys())
```

o/p
True

To check value

```
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print('name' in dictionary.values())
```

o/p True

.item method:
To show all keys and values of items

```
dictionary = {
'player' : 'name',
'power': 'full',
'bonus': '2 lives'
}
print(dictionary.items())
print('name' in dictionary.values())
```

o/p
dict_items([('player', 'name'), ('power', 'full'), ('bonus', '2 lives')])

True

Tuple:
A new data structure.
Tuple is :

- **Immutable:** Not subject to change.
- **separated by comma and enclosed in round brackets.**
- **Objects are orderly arranged.**
- **It is almost similar to list, only difference is in immutability and round brackets)**

```
test_tuple = (1,2,3)
print(test_tuple)
print(test_tuple[0])
```

o/p

(1, 2, 3)
1

Immutability:

```
test_tuple = (1,2,3)
test_tuple[0] = 10
print(test_tuple)
print(test_tuple[0])
```

o/p

n <module>
  test_tuple[0] = 10
TypeError: 'tuple' object does not support item assignmen

Tuple Unpacking

```
test_tuple = (1,2,3)
a,b,c = test_tuple
print(a)
print(b)
print(c)
```

o/p

1
2
3

As Tuple is immutable so it can also be placed in dictionary

```
dictionary={
(1,2,3) : 'name',
'power' : 'full',
'bonus' : '2 lives',
}
print(dictionary[(1,2,3)])
```

o/p
name

Application of "in" method in tuple

```
test_tuple = (1,2,3)
print (1 in test_tuple)
```

O/p
True

```
test_tuple = (1,2,3)
print (100 in test_tuple)
```

o/p
False

Three Important methods which can be called on tuple:
len() : to know total numbers of objects in a tuple.

.count(given value): to check the number-of-time a value is given.

.index(index value): to know the index-number of given value

Note: in both .count and .index method values are given.

```
test_tuple = (1,2,3,4,5,6,7,8,9,5)
#to know total number
print (len(test_tuple))
#how many time 5 is repeated
print(test_tuple.count(5))
#what is the index of 2
print(test_tuple.index(2))
#what is the index of 5. First, occurence will be in the o/p.
print(test_tuple.index(5))
#To know the value at index 1
print(test_tuple[0])
```

o/p
10
2
1
4
1

Set Data Structure
#set is UNORDERED just like dictionary and holds UNIQUE OBJECTS. Its enclosed in {} brackets.

```
#unique objects in a set
test_set = {1,2,3,5}
#unique and similar objects
test_set2 = {1,2,3,5, 5,6,6,7,7}
```

```
print(test_set)
#check out the difference. Here repeated objects are not printed
print(test_set2)
```

o/p

{1, 2, 3, 5}
{1, 2, 3, 5, 6, 7}

```
#.add-Method: to add any value.
test_set = { 2,3,4,5,5,5,6,6,7,7,9 }
test_set.add(100)
print(test_set)
#if you analyse the result. As data set is not adding an object at after 9 but at
arbitrary position. Moreover, as it is unordered it's value at specific index
can't be found.
print(test_set[0])
```

o/p

{2, 3, 4, 5, 6, 7, 100, 9}
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    print(test_set[0])
TypeError: 'set' object is not subscriptable

```
#DATA SET is data structure so that it can store all types of data.
test_set = { 'a',True,3,4,5,5,5,6,6,7,7,9 }
print(test_set)
```

O/P

```
#Intersection, union and difference.
data_set = {1,2,3}
data_set1 = {1,9,7,8,3}
#union is the combination of all
print(data_set.union(data_set1))
#intersection is the common of all
print(data_set.intersection(data_set1))
#values of data_set which are not present in data_set1.
print(data_set.difference(data_set1))
#values of data_set1 which are not present in data_set.
print(data_set1.difference(data_set))
```

o/p

{1, 2, 3, 7, 8, 9}
{1, 3}
{2}
{8, 9, 7}

Pyhton: Conditionals

Sunday, October 18, 2020 12:42 PM

```
bhoot = True
False
if bhoot:
print(bhoot)
else:
print('move right')
```

O/P
True

```
cat = False
mouse = False
owner = True
#when first condition is executed, rest of the code will not be read.
if cat:
print('mouse run away')
elif owner:
print('Mouse scared')
else:
print('mouse eat chease')
```

o/p

Mouse scared

#and:it will return True when both will be true.

#or: it will retun False when only one will be true.

```
cat = False
dog = True
owner = True
if cat and owner:
print('mouse run away')
elif dog or owner:
print('dog here bro')
else:
print('mouse eat chease')
```

O/P
dog here bro

Returning False and true by multiple values

```
#Whatever value is given to "if", it eventually converted into bolean.
cat = "meaon"
dog = "woof"
owner = True
print(bool('meaon'))
print(bool('woof'))
if 'meaon' and 'woof':
print('mouse run away')
elif dog or owner:
print('dog here bro')
else:
print('mouse eat chease')
# "maon" and 'woof' are returning true. So, how it could be decided that which value would return either
```

```
true or false?
```

O/P
True
True
mouse run away

```
#The following values are considered false in Python:
None
False
Zero of any numeric type. For example, 0, 0.0, 0j
Empty sequence. For example, (), [], ''.
Empty mapping. For example, {}
objects of Classes which has __bool__() or __len()__ method which returns 0 or False
All other values except these values are considered true.
```

Logical Operator:
To convert any condition into Bolean. This work is done by Logical Operator.
There are six Logical Operators.

1. Less than (<)

2. Less than or equal (<=)

3. Greater than (>)

4. Greater than or equal (>=)

5. Equal (==)

6. Does not equal (!=)

Why the need of LO?
Base unit of conditional is True & False. Therefore, operators are used to convert any condition into Bolean.

In two conditions of LO if even one is true, result will be true.

```
print( f"1==1 is {1==1}")
print(f"1>1 is {1>1}")
print(1<1)
print(1>=1)
print(1<=1)
print(1!=1)
```

o/p
1==1 is True
1>1 is False
False
True
True
False

Identity Operator

```
#Identity Operator :it compares the memory location (logical operator compares value)
#look at the logical operator given below. here values are being compared.

print([1,2,3] == [1,2,3])

#to compare memory location "is" operator is used. In following example it is checked that whether both
lists are stored in similar location or not.
```

```
print([1,2,3] is [1,2,3])
```

O/P
True
False

Loop and Iterables

```
#for Loop
#Mechanism of for loop :u have shop
mylist = ['apple ','banana ' , 'cherry ', 'eggs ']
#first way
print('Result of first method\n')
print(mylist[0]+ 'price tag')
print(mylist[1]+ 'price tag')
print(mylist[2]+ 'price tag')
print(mylist[3]+ 'price tag\n')

#2nd Way. Use of 'next' funtion after making the list iterator that the list is unable to give us next value by itself .Therefore:

print('Result of 2nd Method \n')
newlist = iter(mylist)
print(next(newlist)+'price tag')
print(next(newlist)+'price tag')
print(next(newlist)+'price tag')
print(next(newlist)+'price tag\n')
#there is a problem in second method of Next function that it will break the code. As there is not any option to contain it automatically or to stop it from moving ahead from the last value. This can be done by for loop.

#3rd Method.
#for variable (in which value is to be stored) in (to convert into iterator) list (whose values are to be iterated)

print('result of third method\n')
for anotherlist in mylist:
  print(anotherlist +'price tag')
```

Note: for loop converts the object into iterator and then run defined functionality on it one by one.

o/p

Result of first method

apple price tag
banana price tag
cherry price tag
eggs price tag

Result of 2nd Method

apple price tag
banana price tag
cherry price tag
eggs price tag

result of third method

apple price tag
banana price tag
cherry price tag
eggs price tag

```
#nestd for loop.
list1 = ['1','2','3']
list2 = ['a', 'b', 'c']
#how print all values of list2 for every value list1 i.e. 1,2 and 3.
for x in list1:
  for y in list2:
  print(x," ", y)
```

o/p

```
1  a
1  b
1  c
2  a
2  b
2  c
3  a
3  b
3  c
```

```
#Take a number and generate next 10 whole numbers using nested for loop
l_1 = [1,2,3]
l_2 = [1,2,3]
x=int(input("enter number: "))
for i in l_1:
for j in l_2:
x=x+1
print(x)
```

o/p

enter number: 5

```
6
7
8
9
10
11
12
13
14
```

#print index, value of x and y in different columns.

```
#nestd for loop.
list1 = ['1','2','3']
list2 = ['a', 'b', 'c']
#how print all values of list2 for every value list1 i.e. 1,2 and 3.
z=0
for x in list1:
  for y in list2:
    z += 1
    print(z," ", x," ", y)
```

o/p

index x-vale y-value

```
1   1    a
2   1    b
3   1    c
4   2    a
```

```
5   2   b
6   2   c
7   3   a
8   3   b
9   3   c


#Exercize table of 2. With single for loop.

x = [1,2,3,4,5,6,7,8,9,10]
for z in x:
y = z*2
print('2 x',z,'=', y)

o/p
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20


--------------------------
```

Another way

```
#Exercize table of 2
x=range(1,11)
for z in x:
y = z*2
print(f"2 x {z} = {y}")
```

o/p

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20


--------------------------

#Exercize table of 2, with nexted for loop.
list1=range(1,11)
list2 = [2]
for x in list1:
  for y in list2:
  print(f"2 x {x} = {y*x}")

o/p

2 x 1 = 2
2 x 2 = 4
```

2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

While Loop

```
#while loop keep on running utrill the condition becomes False.
x = True
while x:
print('i am running ')
```

o/p

i am running
i am running
i am running
i am running
i am running
i am running
i am running

```
#what while loop is actually doing.
x = 2
y = 1
while x:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
#so, it is returning BOLEAN value
```

o/p

i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True
i run because, 2 > 1, returns True

```
x = 10
y = 1
while x>y:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
#so, it is returning BOLEAN value
y+=1
```

o/p

urns True
i run because, 10 > 6, returns True
i run because, 10 > 7, returns True
i run because, 10 > 8, returns True

i run because, 10 > 9, returns True

Else can also be used with While but it will be executed when while would be false.

```
x = 2
y = 0
while x>y:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
y+=1
#now y is more than x so condition has become false. Now, y else will execute
else:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
```

o/p

i run because, 2 > 0, returns True
i run because, 2 > 1, returns True
i run because, 2 > 2, returns False

Continue, Break, Pass

```
#continue, break and pass.
#break: python will stop executing further code in a loop when there will be break.
x = 2
y = 0
while x>y:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
y+=1
#here break is mentioned below to move out of while-else loop.
break
else:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
#after coming out of loop now "hello" will be executed
print('hello')
```

o/p

i run because, 2 > 0, returns True
hello

```
#continue: it will not let the code be read further but keep on restarting.
x = 2
y = 0
while x>y:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
continue
y+=1
#here break is mentioned below to move out of while-else loop.
else:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
```

o/p

i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True

i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True
i run because, 2 > 0, returns True

```
#pass:to skip any error or incomplete program
x = 2
y = 0
while x>y:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
y+=1
#here break is mentioned below to move out of while-else loop.
else:
print(f" i run because, {x} > {y}, returns {bool(x>y)}")
#for loop is not completely defined. So, in order to skip this part from execution "pass " will be used.
for z in range(10):
pass
```

o/p

i run because, 2 > 0, returns True
i run because, 2 > 1, returns True
i run because, 2 > 2, returns False

Functions

Saturday, October 24, 2020 12:07 AM

Functions are used:

- To wrap
- To departmentalize
- DRY: don't repeat yourself

Def function_name():
A function has to be called in order to print/execute it.

```
def fun_name():
print('x')

#printing of x is because of this.
fun_name()
```

o/p
x

```
#parameter
#def adder(parameter):code designer
#argument
#adder(argument):code user

def fun_name(x,y):
print(x+y)
#printing of x is because of this.
fun_name(2,2)
```

o/p
4

```
#parameter
#def fun_name(parameter):code designer

#argument
#fun_name(argument):code user

x = int(input("enter value of x "))

y = int(input('enter value of y '))

#to print type of values

print(type(x))
#defining parameters
def fun_name(x,y):
```

```
print(x+y)
#printing of x is because of this.

#Providing argument. Here taking from 3rd person.
fun_name(x,y)
```

o/p

```
enter value of x 5
enter value of y 5
<class 'int'>
10
```

```
#Positional parameters: whose positions are fixed.
def receipt(name, amount):
print(f"Hello {name}! you paid $ {amount}.")

#Positional arguments: whose position are fixed.
receipt('ali',10)
#as arguments are fixed. Therfore, output is disturbed.
receipt(10, 'ali')
```

o/p

```
Hello ali! you paid $ 10.
Hello 10! you paid $ ali.
```

```
#taking arguments from users:
#Positional parameters: whose positions are fixed.
def receipt(name, amount):
  print(f"Hello {name}! you paid $ {amount}.")

#Positional arguments: whose position are fixed.
name = input('your name: ')
amount = int(input('Enter paid amount: '))
receipt(name, amount)
#as arguments are fixed. Therfore, output is disturbed.
receipt(amount, name)
```

o/p

```
your name: awais
Enter paid amount: 10
Hello awais! you paid $ 10.
Hello 10! you paid $ awais.
```

```
#keword arguments. Using assignment operator directly gives the values.
def receipt(name, amount):
```

```
print(f"Hello {name}! you paid $ {amount}.")
#Look at here. Even arrangement is disturbed but output is same.
receipt(amount = 100, name = 'usman')
receipt(name = 'khan', amount = 150)
```

o/p

Hello usman! you paid $ 100.
Hello khan! you paid $ 150.

#default value: Parameters are already defined in it. If arguments are not added, defined values will be shown.

```
def receipt(name = 'john', amount = 10):
print(f"Hello {name}! you paid $ {amount}.")
#no arguents
receipt()
receipt()
```

o/p
Hello john! you paid $ 10.
Hello john! you paid $ 10.

Return Keyword:

#Return: a a function must return something which means function must give some values. Moreover, it is returned for a single time.

```
def receipt(name = 'john', amount = 10):
print(f"Hello {name}! you paid $ {amount}.")

#here the value is not given by fuction but it is just showing up the fed value.
receipt()
#None, becuase function is not returning and thats why it is unable to store any value in x.
x = receipt()
print(x)
```

o/p

Hello john! you paid $ 10.
Hello john! you paid $ 10.
None

#Return: a a function must return something which means function must give some values.

```
def receipt(name = 'john', amount = 10):
```

```
print(f"Hello {name}! \n you paid $ {amount}.")
#returning the value of function
return(amount)


#now it is returning value and saving in "x"
x = receipt()
tax = 2
print(f" your paid amount after deduction of tax is ${x-2}")
```

o/p

Hello john!
you paid $ 10.
your paid amount after deduction of tax is $8

```
#Doc string: to give information about function to help the code reader. it
is defined whith three double or single qoutes.

def receipt(name = 'john', amount = 10):
"""
receipt: function
name : take name
amount: takes amount
return: amount
"""
print(f"Hello {name}! \n you paid $ {amount}.")
#returning the value of function
return(amount)


#now it is returning value and saving in "x"
x = receipt()
tax = 2
print(f" your paid amount after deduction of tax is ${x-2}\n")
help(receipt)
#another method
print(receipt.__doc__)
```

o/p

Hello john!
you paid $ 10.
your paid amount after deduction of tax is $8

Help on function receipt in module __main__:

receipt(name='john', amount=10)
receipt: function
name : take name
amount: takes amount
return: amount

```
 receipt: function
 name : take name
 amount: takes amount
 return: amount



#WRITE A CODE O COUNT CAPITAL LETTERS AND SMALL LETTERS SEPERATELY
#word_line = 'ABCDefgh'
counter = {'upper_counter' : 0, 'lower_counter' : 0}
#print(f"given alphabets are \n {word_line}")
def count_letters(word_line='FFFFFFF'):
 for get_value in word_line:
  result = get_value.isupper()
  if result == True:
   counter['upper_counter'] +=1
  else:
   counter['lower_counter'] +=1
 print(f" total numbers of capital letters are {counter['upper_counter']}")
 print(f" total numbers of small letters are {counter['lower_counter']}")

count_letters('HHHHHHHHHHaaaaaa')
```

o/p

```
total numbers of capital letters are 10
 total numbers of small letters are 6
```

#if parameters are defined, there must be an argument while calling the function.
#If variable is defined globally, in parameters and also in arguments. Compiler would first take the value defined in argument. If variable is not defined in argument, it would take value from parameters. If it is also not defined in parameters it would take value globally.

#scope of function:
It means it is not compulsory that all variables are accessible from anywhere in program.
#variable within a function is not accessible from outside of the function.

```
#this variable is accessible from anywhere
c= 10
def fun():
 #this c is accessible only from within this funtion.
 c=20
 return c
#this "c" is taking the value of "c" defined globally.
print(c)
print(fun())
```

o/p
10

20

#NESTED FUNCTION:

```
phone = 'iphone10_onRoad'
def parents_home():
 phone = 'iphone_on_parents_home'
 def my_home():
   phone = 'iphone on home'
   return phone
 return my_home()


print(parents_home())
print(phone)

#python starts with linner funtion/local scope(my_home)
#then it looks into outer function/parent scope/non local scope(parents_home)
#Then it looks into global scope.
#then in built in function
```

o/p
iphone on home
iphone10_onRoad

```
#How built_in finction is approached.
phone = 'iphone10_onRoad'
def parents_home():
 phone = 'iphone_on_parents_home'
 def my_home():
   phone = 'iphone on home'
   #HERE max is being returned which is a built in functtion.
   return max
 return my_home()


print(parents_home())
```

o/p
<built-in function max>

Key-Word:
non local: fixed the value of variable to be taken from non-local space, provided, if it is not defined in local space.

Global: fixed the value of variable to be taken from global provided if it is not defined in local

space.

Example of non-local Keyword

```
phone = 'iphone10_onRoad'
def parents_home():
phone = 'iphone_on_parents_home'
def my_home():
nonlocal phone
return phone
return my_home()

print(parents_home())
```

o/p
iphone_on_parents_home

NONLOCAL: it will take the value from non-local space only when variable is not defined in local space.

it firstly checks that whether that variable is defined above it or not within local-space. If it is defined, it will give an error. Moreover, if variable is defined below nonlocal keyword, it will be ineffective and compiler will take the value of variable defined in local-space.

Example of Global Keyword

```
phone = 'iphone10_onRoad'
def parents_home():
phone = 'iphone_on_parents_home'
def my_home():
global phone
return phone
return my_home()

print(parents_home())
```

o/p
iphone10_onRoad

NONLOCAL: it will take the value from global space only when variable is not defined in local space.

it firstly checks that whether that variable is defined above it or not within local-space. If it is defined, it will give an error. Moreover, if variable is defined below nonlocal keyword, it will be ineffective and compiler will take the value of variable defined in local-space **as well as the variable defined globally will also be ineffective** which means compiler would take value from local-space even the globally defined variable is being called. SEE:

```
phone = 'iphone10_onRoad'
def parents_home():
  phone = 'iphone_on_parents_home'
  def my_home():
    global phone
    phone = 'phone is in my home'
```

```
   return phone
 return my_home()


print(parents_home())
print(phone)
print('making global variable ineffective and taking value from local space')
```

O/P

```
phone is in my home
phone is in my home
making global variable ineffective and taking value from local space
```

```
#Exercize: Take list and add 1 in value of every index of list.
li = [1,2,3] #data
def add_fun(data_taker):
  new_li = []
  for value_taker in data_taker:
    new_li.append(value_taker+1)
  return new_li

print(add_fun(li)) #process
print(add_fun(li))
```

o/p
[2, 3, 4]

#map function: in this method map function brings process and data in a single statement.

```
#map(process address, data)
```

It takes the process from given address and apply on every individual value of every index of list.

```
#another way to solve that exercize
li2 = [2,3,4] #data
def adder(para_li):
 return para_li + 1
print(list(map(adder,li2))) #data + process
```

o/p
[3, 4, 5]

With set datastructure:

```
li2 = [2,3,4] #data
def adder(para_li):
  return para_li + 1
print(set(map(adder,li2))) #data + process
```

o/p

{3, 4, 5}

With Tuple data-structure:

```
li2 = [2,3,4] #data
def adder(para_li):
  return para_li + 1
print(tuple(map(adder,li2))) #data + process
```

o/p

(3, 4, 5)

```
#filter: it works as map function but it filters all the values of list
#filter(process address, data)
li2 = [2,3,4,5,6,7]
def odds(para_li):
 return para_li%2 != 0
print(list(filter(odds,li2)))
```

o/p

[3, 5, 7]

```
#USING FILTER removing the name of usman from list
name = ['awais', 'ali', 'usman']
def remover(para_li):
 if para_li != 'usman':
  return para_li
print(list(filter(remover,name)))
```

o/p

['awais', 'ali']

```
#zip function: it takes two or more iterable and combine it.
li_one = [1,2]
li_two = [3,4]
print(list(zip(li_one,li_two)))
```

o/p

[(1, 3), (2, 4)],

```
first_name = ['shahrukh', 'Anglina']
second_name = ['khan','julie']
E_mail = ['sk@gmail.com', 'aj@gmail.com']
print(list(zip(first_name, second_name, E_mail)))
```

o/p

[('shahrukh', 'khan', 'sk@gmail.com'), ('Anglina', 'julie', 'aj@gmail.com')]

Note: indexing must be compatible or in equal ratio.

#reduce: This does not exist in old version of python language. It's a new concept. It has to be imported.
#FUNCTION: It reduces the entries by applying certain action upon it.
from functools import reduce

```
from functools import reduce

rain_water = [1,2,3,4,5]
def collector(bucket,water):
  return bucket + water
#reduce(function_call, list, intial value)
print(reduce(collector,rain_water,0))

print('Result of next code')
rain_water = [1,2,3,4,5]
def collector(bucket,water):
  return bucket + water
print(reduce(collector,rain_water,5))

print('Result of next code')

rain_water = [1,2,3,4,5]
def collector(bucket,water):
  return bucket * water
print(reduce(collector,rain_water,5))
```

o/p

15
Result of next code
20
Result of next code
600

#list comprehension
#creating list:case 1
li_one = []
for i in range(10):
  li_one.
(i + 1)
print(li_one)
#Now creating by List Comprehension method: it has two part [expression & loop]
line_one = [i for i in range(10)]

```
print(li_one)
```

o/p

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

#list comprehension

```
#creating square list:case2
li_one = []
for i in range(10):
  li_one.append(i ** 2 )
print(li_one,'\n')

#Now creating by List Comprehension method: it has two part [expression & loop]

print('Through Second Method\n')

line_two = [j**2 for j in range(10)]
print(line_two)
```

o/p

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Through Second Method

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

#list comprehension

```
#creating square list and removing odd:case3
li_one = []
for i in range(10):
  if i%2 ==0:
   li_one.append(i**2 )
print(li_one, '\n')



#Now creating by List Comprehension method: it has three parts [expression & loop & condition]
print('Through Second Method\n')

line_two = [j**2 for j in range(10) if j%2 ==0]# condition is applied of the value of j about receiving only even numbers from the defined range and give square of it in o/p.
print(line_two)
```

o/p

```
[0, 4, 16, 36, 64]
```

#set Comprehension: name{explanation,loop}
li_one = {i for i in range(10)}
print(li_one)

#dictionary comprehension

#dic = {exp for key,value in dic}
dict = {'x':2,'y':4}

my_dict_comp = {K:V**2 for K,V in dict.items()}

#here dict.items() is taking keys and values from dictionary and assigning it to K and V respectively. After that V**2 IS performing function on values and assigning it to key.
print(my_dict_comp)

o/p
{'x': 4, 'y': 16}

#**module:**
ANY .PY file is module.
Why we need?
Projects are normally composed of very large code. Therefore, different chunks are divided into multiple files.

Create two files with name china.py and germany.py
china.py
```
def body():
  return 'body'
```

germany.py

```
def engine():
  return 'engine'
```

```
from china import body
from germany import engine

def com_car():
  print(f"{body()} and {engine()} make car")

com_car()
```

Python Packages:

Method to organize the code.
Module can saved into folder.
Features:
Modules can be saved into folder. Folder which has modules in it is called package.

Create folders with the name of "east" and "west" and save files china.py and germany.py in it.
How to access:
Using east.china and west.germany

```
from east.china import body
from west.germany import interior
def car():
 print(f" {body()} from china and {interior()} from germany")

car()
```

Another Way:
Firstly, module is imported and then defined functions in these modules is applied
using .method() criteria.

```
from east import china
from west import germany
def car():
print(f" {china.body()} from china and {germany.interior()} from germany")
car()
```
