S.No.	Date	Title	Page No.	Teacher's Sign/Remarks

Page No. 1
Date

BASICS OF PYTHON

* COMMENTING IN PYTHON CODE:
  ↳ GOOD TO ADD REFERENCES TO EXPLAIN YOUR CODE
  ↳ # NOTE → ANYTHING AFTER # WILL NOT EXECUTED

Ex:-
PRINT('HELLO') → HERE THE CODE DOESNT PRINT ANY OUTPUT
PRINT('HELLO') # USING PRINT TO PRT HELLO
OUTPUT → HELLO

*MATH OPERATIONS IN PYTHON:
ADDITION: '+' PRINT(1+2)
SUBTRACTION: '-' PRINT(4-2)
MULTIPLICATION: '*' PRINT(5*6)
DIVISION: '/' PRINT(8/2)
REMAINDER: '%' PRINT(17%3) → 2 REMAINDER (MODULUS)
EXPONENT: '**' PRINT(2**5)
FLOOR DN: '//' PRINT(17//3) → 5 NEAREST INTEGER

* NUMBERS IN PYTHON
1] INT → STORE INTEGERS
2] FLOAT → STORE DECIMAL NUMBERS
TYPE() → WILL TELL YOU TYPE OF NUMBER IN PYTHON. Ex:- TYPE(55) → INT TYPE(4.26) → FLOAT

* STRING & BOOLEAN
1] STR → TO STORE NON-NUMBER & NON-BOOLEAN
Ex:- TYPE('HELLO') → STR.
NOTE: STRINGS HAVE TO BE WRITTEN WITHIN ' '.
2] BOOL → TO STORE TRUE/FALSE
EX: TYPE(TRUE) → BOOL TYPE(FALSE) → BOOL
TYPE('TRUE') → STR
TYPE('FALSE') → STR

* VARIABLES IN PYTHON:
↳ NAMES GIVEN TO DATA
NOT A GOOD PRACTICE VARIABLE NAME WITH UPPER CASE, WITH UNDERSCORE.
Ex:-
a = 10 b = 20 c = 30
z = a+b+c
PRINT(z) = 60 (10+20+30) TYPE(z) = INT
HERE, EQUAL TO SIGN '=' ASSIGNS VALUE TO A VARIABLE AND HENCE IT IS CALLED ASSIGNMENT OPERATOR.
Ex:-
STR1 = 'abcde' TYPE(STR1) = STR.
PRINT(STR1) = abcdc
Ex:-
h = TRUE TYPE(h) = BOOL
PRINT(h) = TRUE
Ex:-
d = 15.5 c = 15 TYPE(f) = FLOAT
f = d+c
PRINT(f) = 30.5 (15.5+15)

NOTE THAT VALUE ASSIGNED TO VARIABLE CAN ALSO BE INPUT FROM END USER.

age = INPUT("ENTER YOUR AGE;")
PRINT("YOUR AGE IS:", AGE)

ENTER YOUR AGE; 30
OUTPUT → YOUR AGE IS; 30

* * IN AND THE EXAMPES ABOVE IT IS SINGLE ASSIGNMENT
* * I.E. ONE AT A TIME.

* UNDERSTANDING TYPECASTING IN PYTHON
Ex:-
a = INPUT("ENTER YOUR HEIGHT:")
PRINT("YOUR HEIGHT:", a)
INPUT → 6 OUTPUT → 6
NOW CHECK TYPE(a) IT WILL BE 'STR'. BUT GENERALLY HEIGHT SHOULD BE FLOAT OR INT.

b = int(INPUT("ENTER YOUR HEIGHT:"))
PRINT("YOUR HEIGHT:" b)
INPUT → 6 OUTPUT → 6 TYPE(b) = INT

∴ TYPECASTING REFERS TO CONVERTING OF OUTPUT TYPE FROM ONE TYPE TO ANOTHER TYPE.

* MULTI ASSIGNMENT IN PYTHON
CONSIDER: a=1 b=2 c=3, HERE WE HAVE USED ASSIGNMENT OPERATOR '=' 3 TIMES. HOWEVER PYTHON ADDONS MULTI ASSIGNMENT.
Ex:-
a, b, c = 1, 2, 3
Sum = a+b+c
PRINT(Sum) = 6 (1+2+3)

a, b, c = 3
sum = a+b+c
PRINT(Sum) = 9 (3+3+3)

* USE CASES IN TYPECASTING

1] INT → FLOAT
2] FLOAT → INT
3] INT → STRING
4] FLOAT → STRING

* LOGICAL OPERATORS IN PYTHON:

EQUAL TO → ==
NOT EQUAL TO → !=
GREATER → >
LESSER → <
GREATER & EQUAL → >=
LESSER & EQUAL → <=

} OUTPUT OF LOGICAL OPERATORS WILL BE TRUE OR FALSE

a=200
b=200
PRINT(a==b)
OUTPUT → TRUE

* UNDERSTANDING STRING IN DEPTH IN PYTHON
STRINGS ARE USED TO STORE CHARACTER
EACH CHARACTER WILL HAVE A REFERENCE → INDEXING
NOTE: IN PYTHON INDEXING STARTS FROM '0' (ZERO)
EX:-

P	Y	T	H	O	N
0	1	2	3	4	5

* UNDERSTANDING LENGTH OF STRING & INDEXING
IN PYTHON WE USE LEN() TO GET THE LENGTH
EX:
LEN('HELLO') = 5 [COUNT OF CHARACTERS]
INDEXING:
HELLO
01234 → COUNT IS 5
NOTE: INDEXING CAN BE IN FWD & REV DIRECTION

P	Y	T	H	O	N
-6	-5	-4	-3	-2	-1

EX:- "_" UNDERSCORE IN THIS EXAMPLE DENOTES SPACE B/W TO CHARACTERS.

Z = A_B_C_D
LEN(Z) = 7
∴ "_" SPACE IS ALSO COUNTED AS ONE CHARACTER.

CONSIDER:

P	Y	T	H	O	N
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

HOW TO ACCESS PARTICULAR CHARACTER FROM PYTHON:
LET a = 'PYTHON'

PRINT(a[0]) ⇒ OUTPUT = 'P'
PRINT(a[-1]) → OUTPUT = 'N'

* UNDERSTANDING SLICING IN PYTHON USING INDEXING
SYNTAX → [FROM : TO+1]

PRINT(a[1:4])
HERE IT MEANS FETCH CHARACTERS FROM "INDEX 1 TO INDEX 3"
OUTPUT → Y T H

PRINT(a[3:]) NOTE: HERE TO+1 IS EMPTY.
OUTPUT → H O N
IF BLANK IT WILL GO TILL END.

PRINT(a[:3]) NOTE: HERE FROM IS EMPTY.
OUTPUT → PYT
IF WANT IT WILL START FROM 'D' INDEX.

* HOW TO PRINT STRINGS IN REQUIRED "CASES":
1] LOWER CASE
2] UPPER CASE
3] TITLE CASE

Ex: a = python code
PRINT(str.lower(a)) → python code
PRINT(str.upper(a)) → PYTHON CODE
PRINT(str.title(a)) → Python Code

* UNDERSTANDING DATA TYPE
A DATA TYPE IS AN ATTRIBUTE WHICH TELLS THE COMPILER HOW THE PROGRAMMER INTENDS TO USE THE DATA.

BASIC DATA TYPES ARE:
1] INT
2] FLOAT → NUMBER
3] STRING
4] BOOLEANS

* UNDERSTANDING DISADVANTAGES OF BASIC DATA TYPES:
1] THEY CANNOT STORE MULTIPLE VALUES IN A SINGLE VARIABLE.
2] THEY CANNOT STORE MULTIPLE VALUES OF DIFFERENT DATA TYPES IN A SINGLE VARIABLE.

* UNDERSTANDING ADVANCED DATA TYPES:
THESE DATA TYPES HELP SOLVING ISSUES THAT SIMPLE/BASIC DATA TYPES HAVE:

1] LIST:
↳ STORES A SEQUENCE OF OBJECTS/CHARACTERS
↳ OBJECTS CAN BE OF DIFFERENT DATA TYPES (BASIC)

Ex:-
L = ["MY NAME", 23, TRUE, 55.3]
NOTE IT COVERS : OUTPUT → ['MY NAME', 23, TRUE, 55.3]
1] ALL DATA TYPES
2] ONE VARIABLE AND MULTIPLE OBJECTS
3] " DENOTED WITHIN SQUARE BRACKETS "
4] INDEXING IS SAME AS STRING
5] SLICING SAME AS STRING

* CONCATINATING IS LIST
Ex:-
a = [1, 2, 3, 4]
b = ['A', 'B', 'C']

a + b = [1, 2, 3, 4, 'A', 'B', 'C']
HERE WE GET ALL THE ELEMENTS FROM LIST a & b.
Ex:-
a = [1, 2, 3, 4]
b = [2, 'A', 4]

a + b = [1, 2, 3, 4, 2, 'A', 4]
HERE IT IS CLEAR THAT ALL VALUES WILL BE STORED EVEN IF 2, 'A' IS REPEATED.

* MANIPULATION OF LIST

l2 = ['PUNE', 'BANGALORE', 200, "HELLO"]

LENGTH OF LIST:

LEN(l2) = 4

DELETING FROM LIST:

"del" SYNTAX

del l2[0] → delete FROM LIST L2 THE OBJECT IN '0' POSITION

OUTPUT → l2 = ['BANGALORE', 200, "HELLO"]

del ALSO WORKS WITH "FROM : TO" VALUES

NOTE: DELETE IS USED ONLY TO DELETE FROM INDEX REFERENCE AND NOT ACTUAL VALUE.

USE OF REMOVE:

l3 = ["HELLO", 2, 3.5, "NAME", 55]

LISTNAME.REMOVE() → SYNTAX

Ex:-

l3.remove(2) → ["HELLO", 3.5, "NAME", 55]

l3.remove('NAME') → ["HELLO", 2, 3.5, 55]

NOTE: REMOVE IS USED WHEN THE ACTUAL OBJECT IS KNOWN AND THE POSITION / INDEX IS NOT KNOWN.

Ex:-

l4 = ["HELLO", 2, 3.5, 'A', 2, "NAME"]

l4.remove(2) → ["HELLO", 3.5, 'A', 2, "NAME"]

NOTE: ONLY FIRST MATCHING ENTRY IS REMOVED.

↳ LIMITATION

USE OF POP:

USE OF 'IN' IN LIST:

'IN' IS USED TO CHECK IF AN ELEMENT EXISTS IN A LIST

l5 = [2, 3, "NAME", 55]

Ex:-

2 in l5 = TRUE

7 in l5 = FALSE

USE OF 'NOT IN' IN LIST

5 NOT IN l5 = TRUE

"NAME" NOT IN l5 = FALSE

HOW TO FETCH MIN/MAX VALUE IN LIST

l6 = [1, 2, 3, 4, 5]

PRINT(MAX(l6)) → 5

PRINT(MIN(l6)) → 1

MIN/MAX IS ONLY FOR NUMERIC DATA [NOT STRINGS]

ADDING TO EXISTING LIST:

1] APPEND

LIST.APPEND() → SYNTAX

l6.APPEND(6) = [1, 2, 3, 4, 5, 6]

NOTE: APPEND WILL ADD THE VALUE AT THE END OF EXISTING LIST AND APPEND CAN ONLY ADD ONE VALUE OR ADD AN OTHER LIST. IT CAN NOT ADD MULTIPLE VALUES.

```
l6 = [1,2,3,4,5]
l6.APPEND(6,7) → ERROR
l6.APPEND[[6,7]) → [1,2,3,4,5,[6,7]]
```

2] EXTEND

```
l6 = [1,2,3,4,5]
l6.extend([6,7,8])
OUTPUT = [1,2,3,4,5,6,7,8)
```

NOTE: EXTEND ALLOWS TO ADD MULTIPLE VALUES BUT ONLY AT THE END OF THE LIST.

3] INSERT

```
l6 = [1,2,3,4,5]
l6.insert(1,8)
```

↳ THIS MEANS INSERT '8' at 1st INDEX

OUTPUT → [1,8,2,3,4,5]

NOTE: INDEX OF '2' NOW SHIFTS TO 2nd INDEX

SORTING A LIST:

THERE ARE 2 WAYS TO SORT THE LIST.

1] LIST NAME.SORT()

2] PRINT(SORTED(LIST NAME))

EX:

```
l7 = [2,5,3,1,4]
l7.sort()
Output = [1,2,3,4,5]
```


IN .SORT(), THE FUNCTION MODIFIES THE ORIGINAL LIST IN ASCENDING ORDER.

EX: l8 = [1,3,5,2,4]

PRINT(SORTED(l8))

OUTPUT = [1,2,3,4,5]

PRINT(l8) = [1,3,5,2,4]

HERE, THE FUNCTION WILL DISPLAY SORTED VALUE WITHOUT MAKING ANY CHANGES TO ORIGINAL LIST.

* SUMMARIZING CONCEPT OF LIST AND DIFFERENT MANIPULATION:

→ LIST IS USED TO ASSIGN MULTIPLE VALUES TO A VARIABLE IN A SEQUENTIAL ORDER.
→ LIST IS ALSO USED WHEN THE MULTIPLE VALUES ARE OF DIFFERENT DATA TYPES [ONE OR ANY].
→ LIST IS ALWAYS DENOTED IN SQUARE BRACKETS
→ INDEXING IN A LIST IS SAME AS IN STRINGS I.E. FWD/BACKWARD OR LEFT-RIGHT/RIGHT LEFT INDEXING IS POSSIBLE
→ CAN FETCH MAX/MIN VALUE IN A LIST USING MIN/MAX()
→ IF AN OBJECT IN A LIST IS TO BE DELETED USING INDEX REF, DELETE(del) IS USED, ELSE WE USE REMOVE FUNCTION
→ APPEND = ADD ONE OBJECT TO A LIST AT THE LAST POSITION
EXTEND = ADD MORE THAN 1 OBJECT TO LIST AT -11- -11-
INSERT = WILL INSERT AN OBJECT AT A FIXED POSITION

2] TUPLES

↳ STORES OBJECTS(MULTIPLE) OF MULTIPLE DATA TYPES LIKE LIST.

↳ INDEXING SIMILAR TO LIST

QUESTION: WHAT IS THE DIFFERENCE?

TUPLE IS IMMUTABLE I.E. CANT BE APPENDED OR EXTENDED OR ANYWAY.

IMP: ROUND BRACKED USED TO DEFINE TUPLE

```
t1 = ("HELLO", 1, 3.5, TRUE, "FALSE")
PRINT(t1) - (— " —)
TYPE(t1) = tuple
```

3] DICTIONARIES

IN THIS DATA TYPE, WE CAN USED STRING OR NUMBER OF OUR CHOICE TO INDEX THE OBJECTS.

HERE WE USE CURLEY BRACKETS.

SYNTAX D1 = {KEY1:VALUE1, KEY2:VALUE2, ...}

HERE KEYS HAVE TO BE UNIQUE

Ex:-

```
D1 = {"NAME": "JACK", "AGE": 25, "CITY": "PUNE"}
PRINT(D1)
  = {'JACK', 25, 'PUNE'}
TYPE(D1)
```

4] SETS

SAME AS LIST, USES CURLEY BRACKETS

NOTE: SET CAN HAVE ONLY UNIQUE VALUES

```
S1 = {1,1,2,2,3,4,5}
PRINT(S1) = {1,2,3,4,5}
```


UNDERSTANDING CONDITIONS & LOOPS IN PYTHON

1] CONDITIONAL STATEMENTS:

a] IF

b] ELSE

c] ELSE-IF

TO TAKE DECISIONS BASED ON SUM LOGIC OR CONDITIONS, IF-ELSE OR ELSE-IF CONDITIONS ARE USED.

NOTE: IF & IF-ELSE, ELSE-IF IS USED BASED ON THE NUMBER OF CONDITIONS.

IF CONDITIONAL STATEMENT:

↳ "IF" GIVEN CONDITION IS TRUE/FALSE THEN.....

Ex:-

IF AGE GREATER THAN 22, ADD ON INSIDE

↳ 1 CONDITION

↳ AGE > 22

↳ ACTION "ADD ON INSIDE"

```
SET AGE = a

IF (a > 22)
    PRINT ("ADD ON INSIDE")
```

OUTPUT: If a = 23 → ADD ON INSIDE

If a = 20 → NO OUTPUT AS FALSE CONDITION IS NOT DEFINED (NOT REQUIRED).

IF-ELSE CONDITIONAL STATEMENT:

THIS IS USED TO DEFINE THE FALSE CONDITION IN THE ABOVE EXAMPLE

P.T.O

```
IF (a>22)
    PRINT("ALLOW INSIDE")
ELSE:
    PRINT("DO NOT ALLOW")
```

OUTPUT

a = 23 → "ALLOW INSIDE"

a = 20 → "DO NOT ALLOW"

"IF .... DO THIS, ELSE .... DO THIS"

CONSIDER THE FOLLOWING:

CONDITIONS

1] IF VALUE OF a > 10 PRINT VALUE GREATER THAN 10

2] IF VALUE OF a < 10 PRINT VALUE LESS THAN 10

3] IF VALUE OF a = 10 PRINT VALUE EQUAL TO 10

NOTE: HERE THERE ARE 3 CONDITIONS, IF OR IF-ELSE WILL NOT SUPPORT 3 CONDITIONS. HENCE WE USE "IF" - "ELSE IF" - ELSE.

```
IF(a<10):
    PRINT("VALUE OF A LESS THAN 10")
ELIF(a>10):
    PRINT("VALUE OF A GREATER THAN 10")
ELSE:
    PRINT("VALUE OF A EQUAL TO 10")
```


USE OF MULTIPLE ELSE-IF CONDITIONAL STATEMENTS

CONSIDER: AGE, QUALIFICATION, MARITIAL STATUS

CONDITION TO GET AUSTRALIA PR IS AS FOLLOWS:

AGE < 35

QUALIFICATION = PG

MARITIAL STATUS = MARRIED

LET AGE = a QUALIFICATION = q MARITALSTATUS = M

```
IF(a<35 AND q == "PG" AND M = MARRIED):
    PRINT("ELIGIBLE OF AUSTRALIA PR")
ELIF(a>35):
    PRINT("AGE TEST FAILED")
ELIF(q == "UG"):
    PRINT("QUALIFICATION TEST FAILED")
ELIF(M == "SINGLE"):
    PRINT("MARINAL STATUS TEST FAILED")
```

2] LOOPS:

IT IS USED TO REPEAT ANY ACTION IN A PIECE OF CODE

LOOPS ARE 2 TYPES

1] FOR LOOP

2] WHILE LOOP

FOR LOOP ALLOWS THE CODE TO BE EXECUTED REPEATEDLY. HERE THE ITERATION TO STOP THE LOOP IS DEFINED

WHILE LOOP ALLOWS THE CODE TO BE EXECUTED REPEATEDLY. HERE THE ITERATION TO STOP THE LOOP IS A BOOLEAN CONDITION.

UNDERSTANING FOR LOOP:

CONSIDER: LIST L WITH 5 NUMBERS.
NOW MULTIPY EACH NUMBER BY 10 AND PRINT THE OUTPUT

```
L = [1,2,3,4,5]
FOR i in L:
    PRINT(10*i)
```

OUTPUT: 10, 20, 30, 40, 50

HERE i WILL PICK UP EACH NUMBER IN THE LIST ONE-BY-ONE, MULTIPY BY 10 AND TERMINATE THE LOOP AND PRINTS THE RESULT.

UNDERSTANDING WHILE LOOP:

CONSIDER COUNT AS C WHICH IS <= 10. FOR EACH COUNT PRINT "HELLO". ONCE THE COUNT IS GREATER THAN CONDITION (FALSE) THE LOOP TERMINATES

```
C = 1
WHILE(C <= 10):  → CONDITION
    PRINT("HELLO") → OUTPUT
    C = C+1 → TO INCREASE COUNT
```

NOW C=1, IT SATISFIES CONDITION C<=10, PRINTS "HELLO", COUNT INCREASES TO 2 (C+1) .... THIS CONTINUES TILL C>10 (FALSE)

* UNDERSTANDING FUNCTIONS:

THERE ARE 2 TYPES OF FUNCTIONS:
1] IN BUILT → AVAILABLE BY DEFAULT [LEN() .. etc]
2] CUSTOM: → WRITTEN TO AVOID REPEATING OF CODE

STRUCTURE OF FUNCTION:
↳ NAME OF FUNCTION → TO CALL THE FUNCTION
↳ DEFINITION OF FUNCTION → PURPOSE

Eg:-

```
def GREETINGS_CALL():
    PRINT("HELLO, WELCOME")
```
} DEFINE & PURPOSE

GREETINGS_CALL() → TO CALL FUNCTION

Eg:-

```
def mySum(a,b):
    c = a+b
    PRINT("SUM EQUAL TO:")
mySum(5,10) → TO CALL mySum FUNCTION
```
} PURPOSE IS TO FIND SUM

LAMDA FUNCTION:

IT IS AN OTHER WAY OF DEFINING FUNCTIONS.
HERE WE USE LAMDA INSTEAD OF "def".

LET US COMPARE

```
def add(x,y):
    Return(x+y)
add(1,3)  = 4
```

```
add = lamda x,y : x+y
add(1,3)
  = 4
```

* UNDERSTANDING SPECIAL FUNCTIONS:
INLINE FUNCTIONS

FILTER:

BASICALLY USED TO FILTER OUT VALUE FROM A LIST AND BASED ON CERTAIN CRITERIA.
NOTE THAT IN A FILTER YOU CAN NOT COMPUTE ANYTHING, THE OUTPUT IS PURELY BOOLEAN BASED.

CONSIDER LIST L = [5, 7, 22, 97, 54, 62, 61, 3248]
CONDITION NUMBER X IN THE LIST WHEN DIVIDED BY 2, REMAINDER MUST BE ZERO.
FILTER OUT SUCH NUMBER AND CREATE NEW LIST AS OUTPUT. NEW LIST = NL

```
L = [5, 7, 22, 97, 54, 62, 61, 3248]
NL = LIST(FILTER(LAMBA x: (x%2 == 0), L))
PRINT(NL)
    = [22, 54, 62, 3248]
```

MAP:

SIMILAR TO FILTER → COMPUTING POSSIBLE
→ BUT DOES NOT FILTER → GIVES OUTPUT.

CONSIDER ML = [0, 10, 0, 30, 1]
MULTIPLY EACH NUMBER IN THE LIST BY 10 AND PRINT OUTPUT AS ML1

```
ML = [0, 10, 0, 30, 1]
ML1 = LIST(MAP(LAMBDA x: x*10, ML))
PRINT(ML1)
    = [0, 100, 0, 30, 10]
```

REDUCER:

ACCUMULATOR:

* IMPORTANT LIBRARY'S IN PYTHON

1] NUMPY : FUNDAMENTAL PACKAGE FOR COMPUTING IN PYTHON. BOTH VECTOR & MATRIX POSSIBLE.
N-DIMENTIONAL ARRAY
FOR NUMPY TO WORK, IT NEEDS TO BE CALLED TO PYTHON KERNEL EVERYTIME SESSION IS RESTARTED

IMPORTING:
IMPORT NUMPY AS NP
NP → SHORT FORM (ALIAS NAME)

NUMPY ARRAY → VECTOR (1D)
→ MATRIX (2D)

CONSIDER A LIST L = [1,2,3]
NOW CONVERT THE LIST TO ARRAY
=> NP. ARRAY(L)
ARRAY([1,2,3])

CONSIDER A LIST OF LIST LL = [[1,2,3], [4,5,6], [7,8,9]]
=> NP. ARRAY(LL)
ARRAY([[1,2,3],
[4,5,6],
[7,8,9]])

UNDERSTANDING NP. ARANGE
SYNTAX NP. ARANGE (FROM, TO+1, STEP VALUE)
NP. ARANGE(0,10,2)
=> ARRAY([0,2,4,6,8])

UNDERSTANDING NP. ZEROS & NP. ONES.

NP. ZEROS IS USED TO CREATE AN ARRAY WITH ZEROS.

NP. ZEROS (10)
ARRAY([0,0,0,0,0,
0,0,0,0,0])

NP. ONES IS USED TO CREATE AN ARRAY WITH ONES
NP. ONES((4,4))
ARRAY([[1,1,1,1]
[1,1,1,1]
[1,1,1,1]
[1,1,1,1]])

UNDERSTANDING NP. LINESPACE

UNDERSTANDING OR IDENTITY MATRIX
IN THIS MATRIX, THE DIAGONAL ARE 1

NP. EYE(5) MEANS 5X5 IDENTICAL MATRIX

=> ([[1,0,0,0,0] * ALWAYS CREATES SQUARE MATRIX *
[0,1,0,0,0]
[0,0,1,0,0]
[0,0,0,1,0]
[0,0,0,0,1]])

Page No. 23
Date

* UNDERSTANDING RANDOMIZATION IN NUMPY

NP. RANDOM. RAND()
NP. RANDOM. RANDN()
NP. RANDOM. RANDINT()

(FROM, TO+1, SPACE OR NUMBER IN ARRAY)

Page No. 24
Date

* UNDERSTANDING NP. RESHAPE

CONSIDER AN ARRAY OF 10 NUMBERS

SA = NP. ARANGE(1,10)
=> ([1,2,3,4,5,6,7,8,9]) ← ANSWER.

SA. RESHAPE(3,3)
=> ARRAY([[1,2,3],
[4,5,6],
[7,8,9]])

AGAIN SA. RESHAPE(*,10)

NOTE: MAKE SURE RxC IS IN SYNC WITH NO OF ITEMS IN THE ARRAY.
EX:- 10 ITEMS IN ARRAY → RxC => 2x5 OR 5x2.
R → ROW C → COLUMN

* UNDERSTANDING MAX(), MIN(), argmin(), argmax(), SHAPE, dtype, .T

- MAX() → MAX VALUE
- MIN() → MIN VALUE
- argmax() → INDEX OF MAX VALUE
- argmin() → INDEX OF MIN VALUE
- shape → NUMBER OF ROWS & COLUMNS
- dtype → WILL GIVE YOU DATA TYPE OF ARRAY
- T → TRANSPOSE ARRAY.

* UNDERSTANDING HOW TO SELECT OBJECTS / GROUP OF OBJECTS FROM ARRAY.

CONSIDER ARRAY A = NP.ARANGE(1, 11)
⇒ ARRAY([1,2,3,4,5,6,7,8,9,10])

TO GET NUMBER AT INDEX 8
A[8] = 9
A[FROM : TO+1]
A[2:5] = ARRAY([3,4,5])
A[[2,5]] = ARRAY([3,6])

~~BROADCASTING ARRAY.~~
* BROADCASTING ARRAY:

* UNDERSTANDING HOW TO INDEX A MATRIX / 2D-ARRAY

m = np.array([50,20,1,23], [26,23,21,32], [76,54,32,10], [98,6,4,3])

array([[50,20,1,23],
[26,23,21,32],
[76,54,32,10],
[98,6,4,3]])

LIST OF LIST → ARRAY MATRIX

m[0,3] → 3rd POSITION IN COLUMN = 23
↳ 0th POSITION OF ROW

SLICING MATRIX:

m[3,:] → 3rd POSITION OF ROW, ALL COLUMNS
→ [98,6,4,3]
or
m[3] → [98,6,4,3]

* CUSTOM INDEXING OF MATRIX:

m[:,(0,3)] → ALL THE ROWS, SPECIFIC COLUMNS
→ [20,23],
[23,32],
[54,10],
[6,3],
[23,50]

* UNDERSTANDING SELECTION TECHNIQUES:

```
sm = np.arange(1,31)
array([1, 2, ..... 30])

sm % 10
array([True, ... Odd 9, rest False])
        BOOL → OUTPUT

a = sm%10
sm[a] = ([1,2,3,4,5,6,7,8,9])
      OR
sm[sm%10]
```

sm + sm → values at same INDEX add up.
sm - sm
sm * sm
sm / sm

10/sm → 10 ÷ by each value in array.
sm+1 → 1 + for ——u——

* UNDERSTANDING UNIVERSAL FUNCTIONS:

NP. SQRT (sm)	FOR ARRAY WITH DECIMAL VALUE
NP. exp(sm)	= sm1
NP. log(sm)	NP. round(sm1)
NP. max(sm)	NP. round(sm1, decimal=2)
NP. min(sm)	NP. std(sm)
NP. Arg max(sm)	– INDEX POSITION OF MAX
NP. Arg min(sm)	– ——u—— – MIN
NP. cos, sin, tan	NP. var(sm)
NP. Square(sm)	NP. mean(sm)
TO GET UNIQUE VALUES OF ARRAY WITH STRINGS → 'S'	NP. unique(s)

* UNDERSTANDING INPUT OUTPUT

```
s = np.arange(0,20)
NP.SAVE('s1', s)
        → NAME = s1
        → WHICH ARRAY = s
```

↳ THIS GETS SAVED IN WORKING DIRECTORY.

```
NP.SAVEZ('s2.npz', a=s, b=s1)
          → NAME
```
MORE THAN 1 ARRAY

```
NP.LOAD('s1.npy')
```
→ LOADS IT BACK INTO PYTHON.

```
ARCHIVE = NP.LOAD('s2.npz')
ARCHIVE[a] → LOADS 's'
ARCHIVE[b] → LOADS 's1'
```

~~NP.SAVETXT('s1.txt', s1, delimiter=',')~~
NP.SAVETXT('s1.txt', s1, delimiter=',')
NP.LOADTXT('s1.txt')

↳ SAVES AS TXT FILE WITH ',' DELIMITER.

DOUBTS:
* BROADCASTING ARRAY
* NP.LINESPACE
* REDUCER
* ACCUMULATOR
* USE OF POP.

* PANDAS AND ITS FUNCTIONS:

VECTOR / ARRAY / MATRIX vs DATAFRAME
↳ ONLY ONE DATA TYPE
↳ MULTIPLE DATA TYPES

VECTOR vs SERIES
↳ NOT POSSIBLE
↳ CUSTOM INDEXING

* FUNCTIONALITIES IN PANDAS

ALWAYS IMPORT
IMPORT NUMPY AS NP } RUN
IMPORT PANDAS AS PD

* CREATE SERIES:

(LIST / ARRAY / DICTIONARY)
↓ CONVERT
PANDA SERIES

1] L = ['A', 'B', 'C', 'D']
L1 = [10, 20, 30]
A = NP.ARRAY([15, 20, 25, 30])
D = {'N': 10, 'X': 30, 'Y': 40}
} LIST, ARRAY, DICTIONARY

PD.SERIES(data = L1)
0 10
1 20
2 30
} → VERTICAL

PD.SERIES(data = L1, INDEX = L)
A 10
B 20
C 30
} CUSTOM INDEX

SYNTAX = PD.SERIES(LIST, LABELS)

PD.SERIES(A)
0 15
1 20
2 25
3 30

PD.SERIES(A, L)
A 15
B 20
C 25
D 30

PD.SERIES(D)
N 10
X 30
Y 40

* INDEXING IN PANDAS DATAFRAME

S = PD.SERIES([1,2,3,4], INDEX = ['C', 'F', 'B', 'G'])
S1 = PD.SERIES([5,6,7,8], INDEX = ['C', 'F', 'Z', 'L'])

S
C 1
F 2
B 3
G 4

S + S1
= C 6
F 8
} COMMON INDEX

B NAN
G NAN
Z NAN
L NAN
} NOT COMMON

↳ CUSTOM INDEX
S['C'] = 1
S[0] = 1
↳ 0 signal underlying INDEX

CREATING DATA FRAME IN PANDAS & UNDERSTANDING INDEXING:

SYNTAX: PD.DATAFRAME()

CONSIDER DATA FRAME WITH 10 ROWS (A,B,C,D,E,F,G,H,I,J) & 5 COLUMNS (S1,S2,S3,S4,S5)

DF['S3'] → ADD ROWS OF S3 COLUMN
DF['S3','S1'] → -ll- OF S3 & S1 -ll-

DF['S6'] = DF['S1'] + DF['S2']
↳ NEW COLUMN IN DF

DF.DROP('S6', AXIS=1, INPLACE=TRUE)
→ TO DELETE S6
→ FOR COLUMN
AXIS=0 → ROWS

NOW DF → S1 ... S5 ONLY
USE INPLACE=TRUE TO MAKE SURE THE COLUMN IS DELETED FROM ORIGINAL DATASET.

DF.DROP('A', AXIS=0)
1ST ROW DROPPED → BUT THIS VIEW

FOR SELECTING ROW:
→ BY LABLE LOCATION
DF.LOC['F']
DF.ILOC[2] → ROW 'C' [0-A, 1-B, 2-C]
2 = INDEX LOCATION

SUBSET OF ROW & COLUMN.
↳ INTERSECTING VALUE FOR A ROW & A COLUMN

DF.LOC['A','S1'] → 1 VALUE

DF.LOC[['A','D'],['S1','S2']] → 4 VALUES.

DF > 10 [TO CHECK IF VALUE > 10]
THE 10×5 DATA BE FILLED WITH BOOLEAN VALUES I.E. TRUE OR FALSE

DF[DF>10]
10×5 → IF VALUE >10 IT WILL BE DISPLAYED
IF VALUE <10 NAAN.

DF[DF['S1']>0.5] FOR A SPECIFIC COLUMN.

DF[DF['S1']>0.5]['S2'] TO GET CORRESPONDING S2 VALUES FOR 'S1' VALUES > 0.5.

USING MULTIPLE CONDITIONS

DF[(DF['S1']>0.5) & (DF['S2']>0)]
↳ OUTPUT SATISFIES BOTH CONDITIONS

INSTEAD OF USING 'S1','S2' ... WE CAN ALSO USE DEFAULT INDEX [0:1] ....

IN DATAFRAME DF
INDEXS ARE A ... J
TO MAKE THIS OR RESET TO ORIGINAL INDEX
IE 0 ... 9.

SYNTAX OF. RESET-INDEX(). HERE THE CUSTOM INDEX BECOMES A COLUMN IN THE DATA FRAME. IF YOU DONT USE INPLACE = TRUE, ONLY VIEW WILL BE CREATED.

ADDING NEW INDEX:
1] DEFINE NEW INDEX.
2] ASSIGN NEW INDEX.

NEWINDEX = ['H', 'I', 'J', 'K', ... 'Q']

DF['NI'] = NEWINDEX
↳ COLUMN NAME FOR NEWINDEX

THE 10X5 DATAFRAME WILL HAVE NEW COLUMN AT THE END WITH NEW INDEX.

MAKING AN EXISTING COLUMN IN DATAFRAME AS INDEX.

DF.SET-INDEX('NI')

NOW 'NI' I.E. H, I, J ... Q WILL BE THE INDEX AND NOT A, B, C ... J.

" PLEASE USES INPLACE = TRUE " IF CHANGE REQUIRED ON DATAFRAME.

HOW TO DEAL WITH MISSING DATA USING PANDAS:

CONSIDER THE FOLLOWING DATAFRAME 'DF'

	BASEBALL	CRICKET	TENNIS
0	5	1	1
1	NaN	2	2
2	NaN	NaN	3
3	5	4	4
4	7	6	5
5	2	7	6
6	4	2	7
7	5	NaN	8

DF.DROPNA() → ROWS 1, 2, 7 WILL BE DROPPED BUT INDEX VALUES DONT GET UPDATED

DF.DROPNA(AXIS=1) → ONLY TENNIS WILL REMAIN.

DF.DROPNA(THRES=2)
THRES 2 MEANS IF ROW HAS ATLEAST 2 VALID RECORDS [ONLY ROW 3 WILL DROP OFF]

TO FILL THE NaN VALUES
DF.FILLNA(VALUE=)

DF['BASEBALL'].FILLNA(VALUE = DF['BASEBALL'].MEAN())
NaN WILL BE FILLED WITH MEAN VALUE.
OR CAL MEAN & ASSIGN
OF VALUE OR COMPUTED VALUE

* AGGREGATING DATA USING GROUPBY:

CONSIDER DATA FRAME 'DF'

	CUSTID	CUSTNAME	PROFIT
0	1001	A	1000
1	1002	B	2000
2	1002	C	3000
3	1002	D	4000
4	1003	E	5000
5	1003	F	6000

df.GROUPBY('CUSTID') = GBCUSTID

GBCUSTID.MEAN() WILL GIVE THE MEAN OF PROFIT FOR EACH 'CUSTID'

.std, .VAR, .MAX, .MIN etc....

.describe()

TO GET FREQ OF OBSERVATION

GBCUSTID.COUNT()

CUSTID	CUSTNAME	PROFIT
1001	2	2
1002	2	2
1003	2	2

* MERGING DATAFRAMES USING PANDAS:

DF1:

	A	B	C	D
0	Y	01	11	A1
1	N	02	13	A2
2	Y	03	19	A3
3	Y	04	16	A4

DF2:

	D	B	C	F
0	C1	01	11	B1
1	C2	02	13	B2
2	C3	03	19	B3
3	C4	04	16	B4

TO CONCAT, SYNTAX PD.CONCAT([DF1, DF2])

HERE COMMON VALUES GET MERGED FOR THE OTHER COLUMNS, IT WILL HAVE NaN VALUES WHERE NOT APPLICABLE. USES AXIS=1 FOR COLUMN FOR COLUMN EACH COLUMN WILL BE TREATED AS UNIQUE COLUMN EVEN IF THEY ARE SAME BUT ROW VALUES WILL BE MERGED IF SAME VALUE FOR EX:- 01, 02, 03, 04 :

* CONCEPT OF MERGE (JOIN ALSO)

PD.MERGE(DF1, DF2, HOW="OUTER", ON="CUSTO.") → SAMPLE MERGE STATEMENT

DIFFERENCE B/W MERGE & JOIN

MERGE → UNIQUE COMMON COLUMN B/W 2 DF'S

JOIN → INDEX B/W 2 DF'S

HOW → LEFT RIGHT OUTER [INNER] → DEFAULT VALUE

OUTER → UNION

INNER → ONLY COMMON

LEFT → ONLY THE LEFT OF OR ABST OF

RIGHT → ONLY THE RIGHT OF OR SECOND OF

DF1.JOIN(DF2) DEFULT IS LEFT JOIN

HOW = CONCEPT SAME AS MERGE

BASIC OPERATIONS IN PANDAS:

df.HEAD() → 1,2.... TOP
df['column'].unique() → Unique values in a column.
.nunique() → no of unique values.
df['column'].value_counts() → counts of unique values.
Sub dataframes with conditions 'and/or'
CAN use custom functions on DATAFRAMES.
—|| → Default functions ||— :
.columns, .index, del, .sort_values(by=' '),
.isnull, .dropna(), .fillna,

df = Pd.read_csv(" .csv") [doc should be in default location]
↳ TO IMPORT CSV

df = Pd.to_csv(" .csv", index=False)
↳ TO Export CSV
→ will not store default index value
=TRUE → index will be first column

df = Pd.read_Excel(" .xlsx", sheet_name=" ")
↳ to import Excel
↳ to which sheet to import.

df = Pd.to_Excel(" .xlsx", sheetname=1:")
↳ TO Export.