Complete PYTHON Handwritten Notes:

Content :

(1). Introduction to Python:

- Why Python ?
- Differences b/w C, Java & Python
- Features of python
- History of python.

Why python ?

- Syntax are very simple.

Syntax = Rules

1. Simple syntax
2. Length of code - Very small
3. Complex problems - Can be solved in a simple manner.

Differences b/w C, C++, Java & Python?

C	C++	Java	Python
- Procedure Oriented	- Object oriented	- object Oriented	Both procedure & object oriented

IDE's for python:

PyCharm, PyDev, IDLE, Jupyter, Notebook, Spyder etc.,

Features of python:

1. Simple to learn
2. Open source → Anyone can use
3. Portability
4. High-level Interpreter → Memory management done itself.
5. object-oriented → by this complex problems also can solve in a simple way.
6. Standard libraries → 200+ (major issue (or) advantage)
   - No. of functions are there in standard libraries
   Ex: Factorial(5) = 120.....
   ..... etc.,
   - In each libraries there are different types of functions are there.

History of python:

- It means no. of versions

V_1 (old) → V_2 (old to new) → V_3 (New to Super new)

Latest version is Version

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

Differences b/w V_2 & V_3:

	V_2	V_3
Input:	print	print()
Output:	Statement	~~Statement~~ function
Ex:	5/2 = 2.0	5/2 = 2.5.

(2). Applications of python:

- We can use (or) write pages like
  - Web & Internet Development
  - Desktop GUI Applications
  - Artificial Intelligence
  - Machine learning
  - Image processing Applications
  - Business Applications
  - Games & 3D graphics
  - Network programming
  - Database Access
  - ...
- These pages are "Research oriented".

(3). Integrated Development Environment: (IDEs)

- These Environments uses to write python programme's.
- Those Environments are
  1. IDLE
  2. Jupyter
  3. Atom
  4. Anaconda
  5. Pycharm
  6. ERIC
  7. Th - Tommy
  8. Spyder ....etc,

→ These 8 softwares are major one.

→ There are so many software programmes.

(4). Introduction to Python Interpreter:

- For how to execute the python programme

Compiler:

Source code (What we write) → Compiler (checks the rules to written programme What we write) → Machine code (If programme rules are correct, then machine code generates. Then we executes the programme) → Output (Here the output comes.)

Interpreter:

Source code → Interpreter → Output

Interpreter ⇓

Compiler → Virtual Machine

Do follow for more inSights

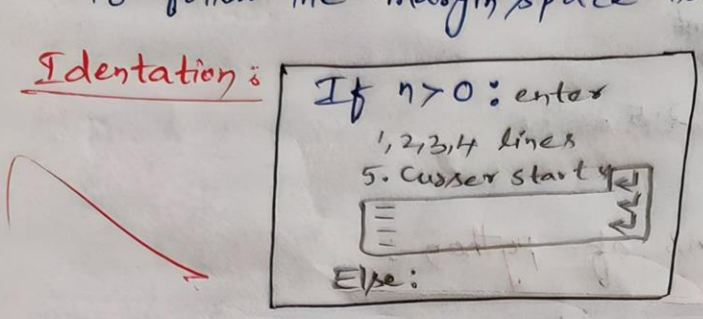
Linkedin: KEDAM MAHESH BABU

Swipe >>>

⑤. Indentation and Comments :

- For to write in "block of statements".
- To represent the block of statements, we need to follow identation.
- To follow the margin space is called Identation.

Identation :



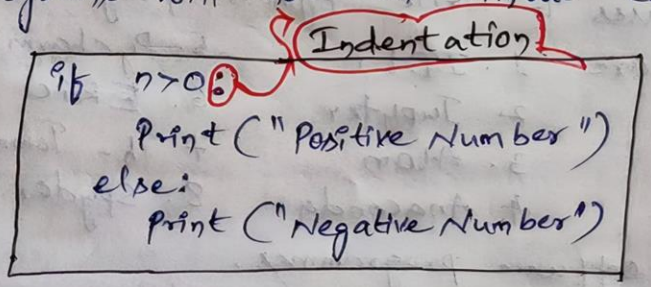
For c : { → Indentation
c++ ; } → " " "

Comments :

Single line comment → # _____ Here we starts with Hashtag # Symbol, then Interpreter doesn't involves in this.

Multi-line comment → ''' _____ ''' (or) """ _____ """

- Python requires identation as a part of syntax.
- Indentation signifies the start and end of block of code.
- Programs will not run without Correct indentation.




```
if n>0
{ print ("positive Number")
}
else
{ print ("Negative Number")
}
```

Single line comment :

as prefer to the line

Multi-line comment :

''' Comment '''

""" Comment """

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

⑥. Key words :

⇒ Keywords are the reserved words which have predefined meaning and functionality.

False	class	finally	is
return	None	Continue	for
lambda	try	True	def
from	global	non local	while
And	del	not	with
as	elif	if	or
yield	assert	else	import
pass	break	except	in
raise.			

⑦. Variables - Identifiers :

Variable name :

- A Variable is defined as an alternate name for memory location and can have a short name or a more descriptive name.

Variable → Name given to memory location.
→ Name should be given by user.
→ User should follow the rules.

Rules for python Variables :

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A Variable name can only have alpha-numeric characters and underscores (A-Z, 0-9, and _)
- Variable names are case-sensitive. (name, Name, NAME are three different variables)
- Variable name should not match with keywords.

Left to right ⇒ Completely Wrong
Right to left ⇒ Correct

Do follow for more inSights
Linkedin: KEDAM MAHESH BABU

Swipe >>>

(8). Built - in - Types (or) Data types :

- These are data types

Numeric :

- Integer → Whole numbers Ex: 3, 400, 5000, etc.,
- Complex numbers → Numbers with real & Imaginary values Ex: a+ib
- Float → Numbers with decimal point. Ex: 3.14, 4.156, ...etc

Dictionary :

- Unordered Key : value pairs Ex: { Key1 : value1, Key2 : value2, ... }

Boolean :

- True or False → Only for checking cases.

Set :

- Unordered collection of unique object. Ex: {10, 20, 30}

Sequence Type :

Strings → Ordered sequence of characters. { ' ', " ", ''' ''' } Ex: 'Python', "Mahesh"

List → Ordered sequence of object. Ex: {10, 20, 30}

Tuple → Ordered immutable sequence of objects. Ex: {10, 20, 30}

- Declaration is not required for python.

Implicit

a = 5 (a: Integer, 5: Integer)

a = 5.2 (5.2: Float)

a = "Welcome" → belongs to strings.

(9). Assigning values to Variables :

1. Basic form :

- This form is the most commonly used form for assignment.

Example : >>> Str = "Hello"

>>> Print (Str)

Hello

Memory location (left side) = Expression (Right side)

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

2. Tuple Assignment:

```
>>> x, y = (50, 100)
>>> print ("x=", x)
    x = 50
>>> Print ("y=", y)
    y = 100
```

3. List Assignment:

- This works in the same way as the tuple assignment.

```
>>> x, y = [3, 5]
>>> Print ('x=', x, "y=", y)
    x = 3    y = 5
```

4. Sequence Assignment:

- Any sequence of names can be assigned to any sequence of values, and Python assigns the items one at a time by position.

```
>>> a, b, c = 'hai'
>>> a
'h'
>>> b
'a'
>>> c
'i'
```

5. Extended sequence unpacking:

- It allows us to be more flexible in how we select portions of a sequence to assign.

```
>>> p, *q = "Hello"
```

- Here, "p" is matched with the first character in the string on the right and "q" with the rest. The starred name (*q) is assigned a list, which collects all items in the sequence not assigned to other names.

```
>>> p
'H'
>>> q
['e', 'l', 'l', o]
```

Here, * means accepting the multiple elements.

6. Multiple target assignment:

- In this form, Python assigns a reference to the same object (the object which is right most) to all the target on the left.

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

```
>>> a = b = 76
>>> Print(a)
 76
>>> Print(b)
 76
```

7. Augmented assignment :

- The augmented assignment is a shorthand assignment that combines an expression and an assignment.

```
>>> x = 8
>>> x += 1
>>> Print(x)
 9
```

> To update something in the assignment is called Augmented.

- There are several other augmented assignment forms.
  forms : -=, **=, %=, etc.,

(10). Input and Output Functions :

> Int = Integer
> Str = String
> ...

Input ( ) :

- The input ( ) function is used to read data from Keyboard.
- It reads data into string format.
- Programmer have to convert data into specific format before using them.
- To convert use the int, float and str functions.
  - Int (2.3) returns 2, int("123") returns 123.
  - Float (6) returns 6.0, float("3.14") returns 3.14.
  - Str (123) returns "123", Str (4.5) returns "4.5"
- If the conversion isn't possible, there will be an error.
  Example : int("abc"), int("two")

Example - 1 : Reading single input without request message.

```
a = input()
Input : 10
```

Example - 2 : Reading single input with request message.

```
a = input("Enter the value of a :")
Input : Enter the value of a
10
```

Example - 3 : Reading multiple inputs in a single line using split ( )

```
a = input().split(" ")
Input : 10  20  30  40  50
```

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

Print( ):

- The print( ) function prints the specified text or value to the screen.

Example - 1 : Print("Hello World")

Output : Hello World.

Example - 2 : Add a new line or vertical space b/w two outputs

```
Print ("Hello")
Print (" Welcome to Python Programming")
```

Output : Hello
Welcome to Python Programming.

Example - 3 : Option keyword argument " sep = "

```
Print ("Welcome", "To", "Python", "Programming")
```

Output : Welcome To Python Programming

```
Print (" Welcome", "To", "Python", "Programming", sep="\n")
```

Output : Welcome
To
Python
Programming.

```
Print ("Welcome", "To", "Python", "Programming", sep=",")
```

Output : Welcome, To, Python, Programming.

Example - 4 : Using keyword argument " end = "

- " end = " is a string appended after the last value, defaults to a new line.
- It allows the programmer to define a custom ending character for each print call other than the default new line or \n.

```
Print ("Hello!", end =" ")
Print ("Welcome to Python Programming")
```

Output : Hello! Welcome to Python Programming.

Example - 5 : Using Format specifiers.

```
a = 10
b = 2.5
Print ("%d is an integer and %f is a float." %(a, b))
```

Output : 10 is an integer and 2.5 is a float.

Example - 6 : Without using Format specifiers.

```
a = 10
b = 2.5
Print (a, " is an integer and", b, " is a float.")
```

Output : 10 is an integer and 2.5 is a float.

(11). Operators :

- Operators are used in "Expressions".

Arithmetic operators → +, −, *, /, //, %, **.
Comparison (or) relational operators → <, >, <=, >=, ==, !=
Assignment operators → =, +=, −=, *=, /=, %=, //=, etc,
Bitwise operators → &, !, ^, ~, <<, >>.
Logical operators → and, or, not
Membership operators → in, not in
Identity operators → is, is not

Arithmetic operators :

a = 3
b = 5 ...

+ → a+b = 8
− → a−b = −2
* → a*b = 15
/ → a/b = 3/5 = 0.6
(modulo) % → a%b = 3%5 = 3
// → Floor Division ⇒ 5//2 = 2 & 5/2 = 2.5 = 2
** → a**b = a^b = 3^5.

% = We need to take remainder
% = modulo

If 5/3 ⇒ 5%3 = 5/3 = 2

Comparison (or) Relational operators :

- Checking condition (or) nothing but comparison.

a == b
a < b
a > b
a != b
a <= b
a >= b

Boolean → Either True (or) False.

Condition
↓
Comparison → Equal → ==
not equal → !=
less than → <
Greater than → >
less than are equal → <=
Greater than are equal → >=

Bitwise operators :

- Convert the given decimal to binary then to complete the operation, after then to convert into decimal.

Decimal → binary → Operation → Decimal.

Ex:

4 bit
a = 5 → 0101
b = 3 → 0011

1 - True
0 - False ...

& (AND)

5 → 0101
3 → 0011
0001 → 1

! (OR)

5 → 0101
3 → 0011
0111 → 7

^ (XOR)

5 → 0101
3 → 0011
0110 → 6

Do follow for more inSights
Linkedin: KEDAM MAHESH BABU

Swipe >>>

~a
~ = complementary
↳ Unary
1 → 0
0 → 1

~a → 0101
1010 → 10 → one's complement

Interpreter makes:
↓
2's complement
~a = − (a+1)
= − (5+1)
= − 6

<< (Left shift operation)
a=5 → 0101
a<<2
a<<2 = 20

8bit
00 000101
000101 00 → 20

>> (Right shift)
0000 01 01
00000 1
↘ 1
a>>2 = 1

Logical operators:

Relational
↓
Condition
a < b
a > b
→ Simple Conditions

a>b log a>c
↓
Compound Condition

Checking the two simple conditions is called "compound condition".

- To know the relation b/w two simple condition then, We use "logical operators".

and
not
or

T – True
F – False

AND
TT → T
TF → F

OR
TF
FT
TT
→ T

NOT
T → F
F → T

- For "and" there should be 2 trues then only we get true, in other cases we get False statement.
- For "Or" Wherever there should one true in the given statement then in all cases we get true statement.
- For "Not" there is no operators.

a>b and a>c

↳ When we use logical operator then we get boolean result.
∴ Boolean means, Either True (or) False

Membership operators:
- These means "search operators".

Ex: We take one sequence list.
Ex: a = [10,20,30]
Result: 10 in a → True (✓)
False (X)

100 in a → True (X)
False (✓)

Do follow for more inSights
Linkedin: KEDAM MAHESH BABU

Swipe >>>

100 not in a → True (✓)
False (✗)

10 not in a → True (✗)
False (✓)

∴ To check the value in membership operators simply we can use "in" (or) "not".

Identity operators : [is, is not]

- This operator should be like "Equality operator."

is → = =
is not → = =

a = 5
b = 5
a = = 5 → True
a is b → True
Id

- Here, we use "Id" as syntax to get the given value.
- Here, result comes based "Id", not based the value.

Ex: id(a) = = id(b) (✓)
a = = b (✗)

∴ Membership & Identity operators are special operators in Python.

(12). Control Structures :

- This structure decides to execute the "Flow of control."
  [What should be next after next.]
- Flow of control should be like
  1. Sequential statement → based on condition which one should be execute.
  2. Conditional statement → based on condition which one should be execute.
     - Here condition means boolean the it comes either True (or) False.
  3. Iterative / Loops statement → A group of statements executing repeatedly (continous)
     → This is also based on condition.
  4. Jumping statement → This will use completely in iterative statements.

Do follow for more inSights
Linkedin: KEDAM MAHESH BABU

Swipe >>>

- Here some statements we should do skip.

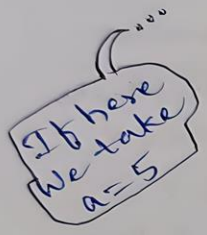
Control structures

- Sequential Execution
- Conditional statements: if, if-else, if elif else, nested if-else
- Iterative statements: while, for, nested loops
- Jumping statements: break, continue, pass.

Conditional statements:

Simple if: if – True block; else – False block. Here only one can execute, not execute both. Here execute goes based on condition.

Condition → if / else




```
if a >= 0:          # Here we need to use Indentation.
    Print ("+ve numbers")   → if it is true.
else:
    Print ("-ve number")   → if it is false.
```

If we take a=5

```
if a>=0:
    Print ("+ve number")   (✓)
else:
    Print ("-ve number")   (✗)
```

If we take a= –5

```
if a>=0:
    Print ("+ve number")   (✗)
else:
    Print ("-ve number")   (✓)
```

- Whenever we use "else" we should not use conditin before it.

  else ← condition (✗)

  elif ← condition (✓).

- In "if" condition again we use "if" condition then it is known as Nested if condition.

Iterative statements:

{ Continously executed.

For loop. While loop.

- Here it stops based on condition.
  - if condition → True (Continous the execution)
  - False (Stops the execution).
- For one execution is called Iteration.
- For five execution is called Five Iterations.
- Here we should remember;
1. Loop Variable should do initialize.
2. Condition should be used to terminate the loop.
3. Loop Variable should be update in every iteration.

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

Syntax's are:

- While condition:
  ≡} statements
  - Here the statements goes continously when it True.
  - Here the statements goes stop when it False.

- Here, same set of statements doing multiple times repeatedly then it is known as Iterative (while) statement.

For loop:

- We can execute ~~for~~ for loop in two methods.

For → Sequential use case execution.
→ Range function use case execution.

Range:

Range (start, stop, step)

- If we don't mention starting value by defaultly it starts with "0" (zero).
- We need to give ending value, it is mandatory.
- If we don't mention difference by defaultly it takes the difference as "1".

Loop Variable	→ Start	→ Starting Value	→ 0	→ Include.
Condition	→ Stop	→ Ending Value	→ Mandatory	→ Exclude.
Updation	→ Step	→ Difference	→ 1	→

Ex: "i" in range (0, 6, 1) (or) range(6)

Start, End, Difference

In 1st Iteration → i = 0
In 2nd Iteration → i = 1
In 3rd Iteration → i = 2
In 4th Iteration → i = 3
In 5th Iteration → i = 4
In 6th Iteration → i = 5

For loop → Definite loop.
While loop → Indefinite loop.

Sequence:

- It means strings, list, tuple we need to take.

Jumping statements:

- Break
- Continue
- Pass

(13). Math & Random modules :

- Here in modules there is a no. of inbuilt functions.

Module → Functions (Built - in)

- codes of math & random are in modules.

Ex: Sin(30°) → Comes directly when we type

Pow(2,2) = 2^2 = 4

⇒ There are different functions in both math & random.

Math module

Random module

Ex: math.Pow(2,2) = 4 (✓)

Pow(2,2) (✗)

Mathematical functions :

- ceil
- copysign(x,y)
- Fabs(x) — absolute
- Factorial(x)
- Floor(x)
- fsum(iterable)
- gcd(x,y)
- Pow(x,y)
- sqrt(x)
- Sin(x)
- Cos(x)
- tan(x)
- Pi
- e → exponential
- tau
- inf
- nan — not a number

ceil means "up"

Ex: math.ceil(10.2)

output : 11

Floor means "down"

Ex: math.floor(10.9)

output : 10

- To execute these mathematical functions, firstly we need to ~~the~~ import modules.

Random functions :

- choice(seq)
- randrange(start, stop, step)
- random()
- Shuffle(list)
- uniform(x,y)

To execute the math & random modules, firstly we need to import them.

- Mainly these random functions are used in games.

Ex: Snake game

Ladder game etc.,

- Here also we need to import random module.

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

(14). Lists : [ ] → Connecting to values.

- Here in lists we get a "multiple values".

Ex: a = 10, 20, 30, 40, 50, 60, ...

(or)

10, 20.5, "hello", 30 → Heterogeneous.

{[ ] ( )} ⇒ Sub-Scripts

- Here lists should be enclosed with square brackets [ ].
- We define list as
  - [Comma separated values]
  - multiple data types.

Execution :

- Creating lists
- Accessing elements from lists
- Slicing
- Reassigning List elements
- Deleting elements
- Multidemensional lists
- Basic operations
  - +, *, len, min, max, sum, membership, iterations.
- List comprehension ⇒ While creating a value in list, based on one formula we can create a list.
- Built-in methods
  1. Append, extend, pop, remove, reverse, len, insert.
  2. Count, copy, sort, index, clear, del.

List Comprehension :
- Here we write code in single line.
- We need to remember the formula here.
  1. Expression.
  2. Iteration.
  3. Condition.

Mutable :

- Once we create a list, we can do modifications whenever we want.

That means,

Elements → We can add / We can delete / We can reassign

Mutable :
List create → modifications
elements { add, delete, reassign }

Immutable :

- Once we create a list, we cannot do modifications.

Immutable :
List create → Can't do modifications

- Here list is mutable object.
- To access the list we can use "index".

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

Index:

Always index starts from '0' (zero)

Forward Index →

Ex:

Index →	0	1	2	3	4
Element (or) Value →	10	20	30	40	50

How can we access (or) use:

Variable.name[index]

a[0] → 10 | a[1] → 20

a[3] → 40 | a[4] → 30

Forward Index →	0	1	2	3	4
Element (or) Value →	10	20	30	40	50
← Backward Index	-5	-4	-3	-2	-1

a[-1] = 50 = a[4]

a[-2] = 40 = a[3]

Slicing:

- Doing extract of some portion with having elements.

Ex:

	0	1	2	3	4
a →	10	20	30	40	50

a[0] = 10

a[1] = 20

⋮

a[4] = 50

a[start:stop]

Include → a[0 : 3] ← Exclude

Input: a[0], a[1], a[2]

Result: [10, 20, 30]

[start : stop]

0 : length of list

len[list]

a[0 : 5] = [10, 20, 30, 40, 50]

0, 1, 2, 3, 4

a[: 3] = 0, 1, 2

a[2 :] = 2, 3, 4

Basic operators:

+ → Concatenation

* → Repetition

Ex: Input: a * 4 → numerical value

List ← Output: [10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

(15). Tuples : ( ) ⇒ Paranthesis

- Tuples are some what different from the list.

Tuple → Immutable
↓
Tuple create
↓
Can't change modifications
X { add, delete, reassign } X

(10, 20, 30)
↳ paranthesis
(Comma separated values)
multiple data type
(10, 20.3, "hello")

a[0] ⇒ [ ]
∴ Here square brackets is called subscript.

- Here, we do accessing from the Index → (0)
  Reverse → -1
  Slicing → [:]

- There is one option in tuple that is

Tuple → Immutable
Firstly convert to list [Tuple]
↳ manipulations
{ add, delete, reassign }
Convert tuple
Here we use tuple Tuple [List] function.

- Tuple's basic operations are as same as in the list.
  Ex: +, *, membership, len(x), min, max, sum, iteration.

methods:

- Methods are
  count → no. of. occurences
  Index → 1st index

Execution:

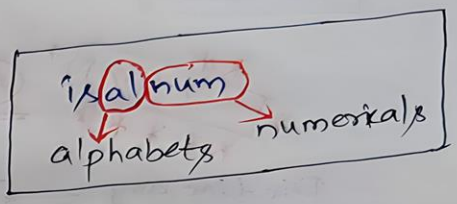
- Create Tuple,
- Accessing elements from tuple,
- Slicing,
- Reassigning Tuple elements,
- Deleting elements,
- Nested tuples
- Basic operations
  +, *, len, max, min, sum, membership, iteration.
- Built-in methods
  count, index.

(16). Strings :

- "Group of characters" is called strings.
- Strings is also similar to lists.
- We will implement the strings in "double Quotations".
- Every character in the double quotations will be taken as "string".

Execution:

- Creating strings,
- Accessing elements from strings,
- Slicing,
- Single, Double, Triple Quotes,
- Format method,
- Basic operations,
  +, *, len, min, max, sum, membership, iterations.
- Built-in methods,
  Capitalize, center, count, endswith, startswith, find, index, rfind, rindex, isalnum, isalpha, isdigit, isspace, islower, isupper, istitle, ljust, rjust, lower, upper, strip, lstrip, rstrip, max, min, replace, split, swapcase, title, zfill.



r - reverse
l - left
r - right

(17). Sets :

- Set → { comma separated values }
  - It accepts multiple data types.
- Here Duplicates will be removed.
- Only unique elements will be in them.
- Elements should be in unorder form.
- It is mutable, Here we can add, delete (or) reassign. (X)
- Here is No Indexing. (X)
  No Slicing. (X)
- How to create Empty set.

empty set - set( )

⇒ Here we need to use set function to create the empty set. ( ) (✓)
{ } (X)

- In set there are different types of operations those are the mathematical functions.
  - Union
  - difference
  - Symmetric difference etc.,

Do follow for more inSights
Linkedin: KEDAM MAHESH BABU

Swipe >>>

Execution:

- Creating sets
- Set as an iterable
- Basic Operations
  add, remove, discard, Pop, clear, len, membership, issubset, issuperset, union, intersection, difference, copy, update, symmetric — difference, intersection — update, difference — update, Symmetric — difference — update

underscore

V.V.Imp

(18). Dictionaries: { }

- Dictionary = { } → {Key1: Value1, Key2: Value2, ...}
  - items
  - Key : Value

Keys → NO Duplicates
Values → Duplicates

Access:

- Here also indexing

Indexing
~~0~~ Key - Index
d[key]

Ex: d = {'a': 2, 'b': 3}
d['a'] = 2

Create:

{ } (✓)
dict (✓)
→ These two creates empty dictionary

- It is the collection of key and value pairs.

Keys are unique.

Execution:

- Creating Dictionary,
- Accessing items from Dictionary,
- Updating Dictionary,
- Reassigning items,
- Deleting items,
- Dictionary Comprehension,
- Built-in methods,
  clear, copy, items, values, keys, update, get, fromkeys, Pop, popitem,

Do follow for more inSights
Linkedin: KEDAM MAHESH BABU

Swipe >>>

(19). Functions:

- Whatever the complex programme is reduced through the functions only.
- Types of functions,
- Function call,
- Function Definition,
- Types of Arguments,
  Required Arguments, Default Arguments,
  Keyword Arguments, Arbitrary Arguments.
- Multiple Returns,
- Local and Global Variables,
- Lambda function,
- Recursion.

Types of functions:

Types
- Built-in functions → Factorial
  Power → Pow
  Squareroot → Sqrt
  Sin
  Cos
- User defined functions
- Lambda functions → Anonymous function.
- Recursion.

User defined functions:

- Here should be function declaration.
- Here should be function call.
- Here should be function definition.

** Function declaration is no need in Python.

- But we need to firstly, write the user defined function.

User defined function → top
User defined function → function call
User defined function → def keyword

- User defined function, we need to start with "def" keyword.

Required Arguments:

- No. of. Parameters should match in both function call & definition.
- Position Arguments also should match.

Do follow for more inSights
Linkedin: KEDAM MAHESH BABU

Swipe >>>

Default Arguments:

- No.of. Parameters may not be equal in both function call & definition.
- Default Arguments should be listed at the end of the parameters.

KeyWord Arguments:

- ~~Here in func~~
- No.of. parameters should be match (or) not match.
- Here we need to use "Variable names".
  - Variable names are called "Keyword".
- No need to follow the position Arguments.

Arbitrary Arguments:

- Here in function call we need to pass "any no.of.arguments".

> Function call → Any no.of. arguments.

- Here in function definition we need to ~~pass~~ accept "only one argument".

> Function def → only one argument → * followed by Variable name.

Multiple Returns:

- Here one function should be accepted by multiple results (Values).

Local & Global Variables:

- Local Variable should used within the function only.
- Global Variable we can use anywhere. [function declaration should be on top].

Lambda function:

- This function can be started always with the word "Lambda".
- This function is also known as "Anonymous function".

Recursion:

- It is like iteration "calling function itself".

Ex:

```
def display ( ):
    display ( )
    display ( ) → function
```

> Here, there is also a infinite recursion such as like iteration.

- Calling the same function is called "Recursion".
- Here, there is also a infinite recursion such as like iteration.
- To stop infinite recursion we need to write "base condition", When base condition is true then recursion immediately turns into terminate

```
def display( ):
    display( )
    display( )
```

Infinite recursion → terminate } × Base condition / recursive call

- For this is best example

Factorial:

```
def fact(n):
    if n==0 or n==1
    return 1
else
return n * fact(n-1) → Recursive cell
fact(4)
```

4 → 4×3×2×1

fact(4)
4 * fact(3)
3 * fact(2)
2 * fact(1)

- In recursive function there is base condition as well as recursive call.

(20). Files:

- Files ⇒ In files we can store the output
  Generally we can't store output anywhere.

* Files → output store → Read

Process to file:

- Open the file
- Read / Write
- close

How to open

open("filename", "mode")

"filename" → Text files
"mode" → Read → only read
Write
W → Write
a → append } write
read
0/1 ← Binary files

fo = file object

∴ fo = open("filename", "mode")

File creation:

In Write mode, In Append mode → File ~~create~~ will be ~~happend~~ Created.

In Read mode, file doesn't created.

∴ fo = open("hello.txt", "w")

Do follow for more inSights

Linkedin: KEDAM MAHESH BABU

Swipe >>>

- What is file ?
- Opening a file
- Access modes
- Closing file
- Writing Data into file
  - Write, Writelines
- Reading Data from file
  - Read, readline, readlines
- File Handling functions
  - Tell, Seek.
- Usage of ~~with~~ WITH Keyword in files

r+ → Reading & Writing
w+ → Writing & reading
a+ → Appending & reading

(21). Libraries in Python :

- Data science :
  - Pandas, Numpy, scipy, scrapy, Matplotlib, Seaborn, Scikit-learn, Tensorflow, scikit-Image, Librosa.
- Data visualization :
  - Matplotlib, ggplot, Altair, Seaborn, Bokeh, Pygal.

Data Manipulation :
- Pandas, Numpy

Database Access :
- SQLAlchemy, Quandl,

Data Modeling :
- TensorFlow, PYTorch, Scikit-Learn, Keras

Web Development :
- Django, Bottle, Pyramid, Turbo Gears, Webapp, Grok, Flask, CherryPy, Hug, Falcon.

Gaming :
- PyGame

GUI Framework :
- TKinter, Wxpython, PYQt5

Robotics:

- PyRobot, DART, SOFA, Pyro

Network Programming:

- Network

Natural language Processing:

- NLTK, scikit-Learn, Polyglot, genism, spacy, rasaNLU.

Deep Learning:

- TensorFlow, Torch, caffe, Theano, Deepmat, ML.NET, Neon, microsoft CNTK, Deeplearning4j

Image processing:

- OpenCV, Skikit-Image, scikit-Learn, scipy, Numpy.

AI & Machine Learning:

scikit-Learn, Theano, TensorFlow, Keras.

Like the post

Follow me on

For more Insights

LinkedIn

@ KEDAM MAHESH BABU

If you find this Helpful

LIKE | COMMENT | SHARE (REPOST)

Thank you !...