

PYTHON PROGRAMMING

→ Adv / features of python :-

- * High level p. lang.
- * Interpreted p. lang. → execute line by line b. code.
- * Highly portable → work any platform / software
- * Extensible. → combine module of other lang.
- * Dynamically typing → assume the type of datatype.
- * GUI (p) and databases support → coded to many libraries
- * Cross platform
- * Support other lang.
- * Easy to code and Read.
- * open source and free → which means that any one can create and contribute to its development.
- * object oriented approach

→ IDLE (Integrated Development and Learning environment).

Has 2 types -

- (1) Shell window
- (2) Editor window

→ Menus →

File menu →

- (1) New file
- (2) Open
- (3) Recent files
- (4) Open module
- (5) Class browser → Shows ()'s, classes and methods in the current editor file / shell
- (6) Path browser → Shows sys.path directories, modules, ()'s, classes and methods in a pre str.

(m) Save

(n) Save as

(o) Save copy as

(p) Print

(q) Close

(r) Exit

IV Edit menu :-

(i) Undo → Undo the last change.

(j) Redo → Redo the last undone change.

(k) Cut

(l) Copy

(m) Paste

(n) Select all

(o) Find in files

(p) Replace

(q) Format menu

V Format menu :-

(1) Intend regions → Intend 4 spaces. → [intend]

(2) Shift selected lines right by "n" spaces.

(3) Dedent regions → Shift selected lines left by the intend width. (default 4 spaces)

(4) Comment out region → Insert # # # #

(5) Un comment region → Remove leading # # # # from selected lines.

(6) New intend width → open a dialogue to change intend width.

VI Run menu :- (editor window only)

(1) Run module → To check module, if no error. Restart the shell to clean the environment, execute the module, output is displayed in the shell windows.

(2) Check module → Check the syntax of module. Open in the editor windows.

(3) Python Shell → open [wakeup] the python shell windows.

VII Shell menu → [Shell window].

(1) View last history :-

Scroll up in shell window to the last shell restart.

(2) Restart shell :- restart the shell to clean the environment.

(3) Previous history.

(4) New history :- cycle through latest entry in history which match the current entry.

(5) Interrupt execution :- stop the running program.

* Debug :- when activated code entered in shell / run from an editor will run under the debugger. [shell window].

* Python prints in os → print("Hello world")

this is a cmd. → online print.

used to prevent execution in testing code. In Python it indicates a cont.

This is a test }
print("Hello") } count.

~~most likely~~ Create → Document

- ① To add a multi-line comment insert a `#` for each line
eg → `# comment`
- ② You can use a multi-line string
eg → `"Hello"` .
- ③ You can use a multi-line string
eg → `'''` more just 1 line
`# comment`

Print ("Hello")

→ Program identifier :-
It is a name given to program entities like class, variables, etc.
Rules →
- Identifiers can be a combination of

- Name of a variable / identifier can have letters, digits / _
- An identifier cannot begin with a digit.
- Identifiers cannot have space & characters like !, @, %, ^, & etc.
- Python keywords cannot be used as identifiers.
- Identifiers can be of any length.

→ Python keywords :-

(1) ~~print~~

→ There are 35 keywords in Python 3.8.

→ These are reserved words which are pre-defined and have a specific meaning.

→ e.g. for, while, def, print, if, break, continue, int, float, True, False, ...

It is a name (identifier) that is associated with a value. As the name implies, a variable is something which can change and therefore can assign different values during execution.

$x = 'Abhishek'$ → String type

$x = 'Jan'$ → String type

$\text{print}(x)$ → Jan

Rules →

A variable name must start with a letter, character.

A variable name cannot start with a no ..

A variable name can only contain alpha numeric characters. (A-z, 0-9)

* Variable names are case sensitive (age, Age, AGE)

code :- different variables

e.g. `x = "awesome"`

`print("python " + x)`

Python print statement is often used to output variable.

To combine both text and a variable, Python uses the '+' character.

→ Datatypes :- which type of data are

stored in a memory location.

It can be many types →

- 1) Numeric → int, float, complex.
- 2) String → str
- 3) Boolean → bool.
- 4) Sequence type → list, tuple, range.
- 5) set type → set
- 6) binary type → bytes array, byte array.

→ Type () :-

You can get the datatype of any object by using type() .

e.g. `x = 5`

`print(type(x))` → `int`

`x = 5j` → `complex`

→ Boolean type () :- (T or F.)

Boolean request 2 values i.e T or F.

e.g. `print(10 > 9) → T`

`print(10 == 9) → F`

⇒ bytes type :-

bytes array

(1) Bytes Data type :- It represents a grp of

byte now just like a array does.

* A byte num is any true integer from 0 to 255.

bytes array end

eg. `elements = [10, 20, 0, 10, 15]`

`x = bytes(elements)`

`print(x[0])` → `10`

* we cannot modify / edit any element in

the bytes type array.

elements = [10, 20, 0, 10, 15]

`x = bytes(elements)`

for i in x:

`print(i)` → ~~10, 20, 0, 10, 15~~

`print(bool(15))` → `T` → because 15 is True
`print(bool("Hello"))` → `T` → str → `True`

`print(bool())` → `F`.

* Python evaluates almost any value to T if it has some sort of content.
* Any string is T, except empty strings.
* Any num is T, except 0.
* Any list, tuple, set and dict are T.
* Any tuple, set and dict are T,
except empty once.

(2) Byte array :-

* Similar to bytes data type, the difference is that the bytes type array cannot be modified by byte [] type can be modified.

* eg → elements = [10, 20, 0, 40, 15]

x = bytes([element])

print(x[0]) → 10

exit {
x[0] = 55 } elements replaced

x[0] = 99

for i in x:

print(i) → 55

E = [10, 20, 30]

x = bytes(E)

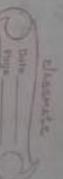
x[0] = 100

print(x[0]) → 100

* List Data types :- []

- * list in Python are similar to []'s in C / Java.
- * list represents ~~the~~ a grp of elements.
- * the main difference b/w a list and an array is that a list can store different types of elements but an [] can store only 1 type of elements.
- * list can grow dynamically in memory but the size of [] is fixed if they [size of [] → fixed] can grow at run time.
- * list → change

Storage multiple type of data
↳ non-mutable
↳ mutable



* Tuple Data type :- ()

- * A tuple is similar to list, it contains a grp of elements which can be different type.
- * The elements in the tuple are represented by commas & enclosed in ().
- * where as list elements can be modified (mutable) but tuple cannot be modified (immutable).

* eg → t1 = (10, 20, 30, 'Python')

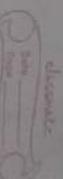
t1[0] = 99 → error (immutable)

* Range Data type :-

- * range data type represents sequence of numbers.
- * the numbers in the range are not modifiable.
- * used for repeating a for loop for a specific number of times.
- * eg → x = range(10)
for i in x:
print(i) → 0
1
2
3
4
5
6
7
8
9

Set → modified
Frozen set → no modification.

classmate



classmate

* Set Data types :- { } (No repeated values)

* Unordered collection of elements → Set.
not fixed preparted sets.

eg → $s = \{10, 20, 30\}$

Store 10, 20, 30

* Set is an unorderd collection of elements ~~much like list~~ in memory.
true order of elements is not maintained in the Sets.

* A set does not accept duplicate elements.

* 2 types → 1) Set

2) Frozen set.

=> Operators :- (o) (operations & aspects.)

- 1) Arithmetic (o)
2) Assignment (o) $+=$, $-=$, $*=$, $/=$
- 3) Comparison (o) (unidirectional logic) $=$, $<$, $>$, \leq , \geq
- 4) Logical (o) AND, OR, NOT
- 5) Identity (o) is , is not
- 6) Membership (o) in , not in
- 7) Bitwise (o) (AND, OR, NOT, SHIFT).

(Output can be comma separated values.)

* update () → used to add elements

into a set. [can be added at any position]

eg → $s = \{10, 20, 30, 30\}$
 $s.append(50, 60)$
 $print(s) \rightarrow \{10, 20, 30, 50, 60\}$.

2) is not → evaluates to false if the variables otherwise.

* frozen set () :- It is same as the set data type, the main difference is that elements in the set datatype can be modified whereas, elements of frozen set cannot be modified.

eg → $s = \{50, 60, 70, 80, 90\}$

$f = \text{frozenset}(s)$

$f \rightarrow \{50, 60, 70, 80, 90\}$

* frozen set () :- It is same as the set data type, the main difference is that elements in the set

datatype can be modified whereas, elements of frozen set cannot be modified.

eg → $s = \{50, 60, 70, 80, 90\}$

$f = \text{frozenset}(s)$

$f \rightarrow \{50, 60, 70, 80, 90\}$

`id()` → memory location



Variables on the either side of the operator point to the same object so there is no change.

Eg → `a = 10`

`b = 20`

`c = a` → Identity (o)

`print(bool(id(a) == id(b)))` → False

`print(bool(id(a) == id(c)))` → True

* Membership (o) :-

Python's membership (o) test for membership in a sequence such as strings, list & tuples.

) `in` → evaluates to true if it finds a variable in the specified sequence if false otherwise.

2) not in → evaluates to if it does not find a variable in the specified sequence if false otherwise.

/* Input method :-

Used input *
Python provides a built-in () calling input function, that gets input from the keyboard.

→ ~~control statements~~

* Statement & expression :-

A statement is a unit of code that the Python interpreter can execute.

• There are different types of statements in Python, → assignment (s), return (s), break, for, while, it import, continue, etc.

• It can be extended to multiple lines using (), [], ;, etc.

• \ - is used for continue the (s).

Syntax →

```
x = input("Enter a name")
print("Hello ", +x)
```

Output → Hello enter a name.

* Type () :-

It returns the type of the specified object.
It is mostly used for debugging purpose.

~~Syntax~~ Syntax → type.(exp).

Eg → count =

```
print("type of count is", type(count))
```

Control Statements.

* control statement is a statement that determines the control flow of a set of instructions.

There are 3 fundamental forms of control that programming lang provides -

- 1) Sequential control
- 2) Selection control → if-else, if, else
- 3) Iterative control → while loop, for loop.

1) Sequential control :- It is an implicit form of control in which instructions are executed in the order that they are return.

2) Selection control :- It is provided by a control statement that selectively execute instructions.

3) Iterative control :- It is provided by an iterative control statement that repeatedly executes instructions.

* Control Stmt :- collectively a set of instructions & control statement controlling their execution → C. Stmt.

* Boolean Expressions :- It is a exp that evaluate to one of 2 boolean

values T | F.

eg → print (a>b) → T | F.

* Comprehend boolean exp :-
Syntax Single logical exp constructed
by using comparison operators can
be chained with true logic / boolean
operators AND / OR / NOT.
eg → if (a>b and b>c)
if (a>b or b>c)

* Selection control statements (Decision
making Statement) → if, if-else, elif.

1) if Statement →

Syntax → if expression:
Statement(s).

eg →
num = int (input ("Enter a number:"))
if (num % 2 == 0):
 print ("num", "is a even")

2) if-else →
Syntax → if exp:
else:

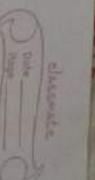
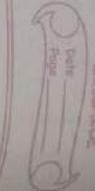
eg →
if num = int (input ("Enter a number:"))
if (num % 2 == 0):
 print ("num", "is a even")
else:
 print ("invalid")

3) Nested if-else →
Syntax → if condition c₁:
(S₁)
 if condition c₂:
(S₂)
 else:
(S₃)

eg → n = int (input ("Enter no:"))
if (n % 2 == 0):
 print ("num", "even")
else:
 print ("num", "odd").

3) Multilevel Selection :-
if - elif - else

Syntax → if exp:
(S₁)
elif exp
(S₂)
else:
(S₃)



* Iterative Control Statement → for, while

It is control statement providing the repeated execution of a set of instructions.

2 types of iteration →

(1) Definite (I) (2) Indefinite (T)

- * Here the no. of times the loop block will be executed is specified explicitly at the time the loop Start.
- * eg → displaying "from 1 to 100." → displaying now from 1 onwards

and asking the user whether the user want to display the int.

I while statement :-

Syntax → while exp:

```
    n=1
    while(n<100):
        print(n)
        n=n+1.
```

→ 1 2 3 99

II for statement :-

Syntax → for i in range(n):

statements.

Syntax → for variable in iterable:

Statement

in → for loop keyword

eg → fruits = ["Mango", "orange"]
for i in fruits:
 print(i)
→ Mango
orange

→ here i is called a loop counter

i.e. n → no. of times that the loop will execute the statement.

⇒ * range () :-

It is a built-in () in python, The range() generates a sequence of integers starting at a . It uses the value by 1 until it reaches n .

So the range() generates a sequence of no. 0, 1, 2, ..., (n-1).

Syntax → for i in range(n):
 print(i).

eg → for *a* in range(5):

print(*a*)

[Default range index no is 0] →

0
1
2
3
4

In this syntax, the range() uses the start the value by 1 until it reaches the stop value.

eg → for *i* in range(5, 10):

print(*i*)

→ 5
6
7
8
9

* range(*start, stop, step*) → range syntax.

eg → for *ak* in range(5, 10, 2):

print(*ak*)

→ 5
7

⇒ 1) Infinite loops :-

It executes the block of statements repeatedly until the user

chooses forces the program to quit.

eg → web server may need to continuously check for incoming connections.

while True:

 print("AK_programz")

To stop this, use break or return

statements. can sys.exit [import sys...]
embedded somewhere within its body. OR
Ctrl + C

In this form you can specify the value that the range() should use.

→ Nested loop :-

for, while

(1) 2-level nested for loop → for loop

eg → for *var* in *seq*:
 for *jav* in *ak*:

Statement(s)

(2) 2-level nested while loop → while loop

eg → while *exp*:
 while *exp*:

Statement(s)

* Delete :- del keyword [entire str deleted]
single char cannot be deleted.

you cannot del / remove char from a str, all the str entirely is possible using the keyword del.

eg = a = 'hello'
print(a)

del a = name error.
print(ex)

* Escape characters :-

It is a backslash (\) followed by the character you want to ins. escape sequences allow for including special characters into str.

eg - newline → \n
a backslash → \\
\b → backspace
\t → horizontal tab
\v → vertical tab

* String operators :-

+ , * , [] , [:] , in , not in
l concatenation slice range slice
add values gives character
of strings str, concat from given
inside of str. gives multiple
copies of the same str.

a = 'Hello' + 'World'

a * 2

a[0] → h

a[1:3] → ello

a[1:3] → ello

→ gives the character from given range.
in → returning true if a character exist in the given str.

a = 'Hello'
print('h' in a) → true

'\n' → returning true if a character not in → returning true if a character does not exist in given str.
print('d' not in a) → true.

* Build-in str methods →

1) Capitalize() → eg → a='ansay'
print(a.capitalize()) → ansay

returning a str whose the 1st character is upper case

Syntax → string.capitalize()

2) Center() → It will center align the str using a specified character (space is default)

Syntax → str.center(length, character)

15 → 16

classmate

classmate

0 6

- * A str is considered a valid identifier if it contains alphanumeric letters ($A-Z$, $a-z$, $0-9$) or _.
- * A valid identifier cannot start with a non alphanumeric letter.

8) islower() :-
Syntax → str.islower()

9) islower() :-
This method returns true if all characters are in lower case.
Syntax → str.islower()

10) isnumeric() :-
This method returns true if all characters are alphanumeric numbers. (0-9, plus full stop)

str → str.isnumeric()

11) isspace() :-

The isspace method returns true if all the characters in string are whitespaces.

obj :- str.isspace()
str.isspace()

12) append() :-
Syntax → list.append(obj).

13) isdigit() → returning true if all characters are digits otherwise false.
Syntax → str.isdigit()

14) identifer → returning true if str is a valid identifier, otherwise f.

l → list

eg → l = ['Ak programz', 'Ak coding']

l.append('Ak buildings')

print(l)

2) insert() :- append() only works for addition of elements at the end of list.

for addition of element at desired position, insert() is used.

Unlike append() which takes 'only' argument, insert() requires 2 args (pos,value) pos→ position.

Rant → list.insert(pos, value)

pos → now specifying in which position to insert the value.

value → specified arg & can be element as any type.(str,no.)

eg → l = [1, 2, 3, 4]

print("Akash", l)

l.insert(3, 2)

print(l)

→ 1 2 3 2
→ 1 2 3 4
→ 1 2 3 12 4

3) count() :- returns count of how many times obj occurs in list

Counter → l.count(obj)

eg → print("Ak", 3) → AkAkAk

t →



classmate

eg = l = [123, 'Ak', 9, 'Ak', 'c', 'Ak']

print(l.count('Ak'))

4) list extend() :- It appends the contents of sequence to l.

l.extend(gro)

l → [] of elements

l = ['phy', 'che', 'bio']

l_2 = list(range(5))

l_2.extend(l) → phy, che, bio, 0, 1, 2, 3, 4

→ Tuple() :-

* Basic tuple operations :-

1) length :-

eg → th = ("apple", "banana", "cherry")

print(len(th))

2) concatenation :-

You can use '+' operator to combine

2 tuples → .(c).

eg → print((1, 2, 3) + (4, 5, 6)) → 1, 2, 3, 4, 5, 6

3) Repetition :-

repeat operator (*) for repeating the elements of a given no. of times using the * operator.

2) Deleting a tuple :-

The elements of a tuple cannot be changed once it has been assigned. That also means you cannot remove or delete an item from a tuple. But deleting a tuple entirely is possible using the keyword "del".

eg = $Ak = ("programz", "bk")$

~~print(Ak)~~
~~del(Ak)~~

~~del(Ak)~~

3) Count() :- returning no. of times a

specified value occurs in a tuple.

Syntax \rightarrow tuple.count(value)

eg $\rightarrow t = (1, 3, 7, 7)$

print(t.count(7)) $\rightarrow \underline{\underline{2}}$

4) index() :- finds the 1st occurrence of the specified values.

It raises an exception if the value is not found.

Syntax \rightarrow tuple.index(value)

eg $\rightarrow t = (1, 3, 7, 8, 7, 5)$

print(t.index(8)) $\rightarrow \underline{\underline{3}}$

[10, 20, "hi"] → C_{10, 10, 20} → file.

C_{10, 10, 20}

03: Strings, lists & tuples

String operator → = [assessing]

* slicing →

Accessing Substring

print("Hello, world")

b = "Hello, world"

print(b[2:5])

String is immutable.

→ Strings in Python :-

a = "Hello world"

print(a[0]). → ~~c~~

- * You can get between a range of characters by using the slice operator.
- * Specifying the start index & end index, separated by a colon, to return a part of string → slicing.

b = "ak_23456789"

print(b[0:10]) → ak_programz.

→ Delete / update a str :-

- * Try to access a character out of index range will raise an error.

- * Index must be a int expressions use float / other type will result in type error.

- * python allows -ve indexing for its sequences.

* Index of -1 refers to the last item, -2 to and last item so on.

ak = "Hello, world",

print(ak[-5:-2]) → ~~olleH~~

→ ~~olleH~~

ak [-5 : -2] → ~~olleH~~

old

b = "ansaybb"

print(b[2:6:2])

→ ~~ansaybb~~

→ ~~ansaybb~~

a = "Hello."

print([:]) → hello.

→ ~~hello~~

* del → strings are deleted.

→ Delete / update a str :-

- * Strings are immutable → means that elements of a str cannot be changed, once it has been assigned.

ak = 'Hello'

print(ak[0])

ak[0] = 'h'

print(ak) → type error.

Delete :- ~~def~~ keyword [entire by deleted].
Example : char cannot be deleted.

You cannot delete / remove the entire file system, deleting the file entirely is possible using the keyword delete.

```
a = hello  
print (txt)  
    ^  
    del txt  
print (txt)
```

Escape characters :- It is a back slash (\) followed by the character you want escape sequences allowed for including special characters into str.

\v → single slash
\b → backspace
\t → horizontal tab
\v → vertical tab

* String operators :-

[E] + → a = 'hello' * → a = 'hello'
b = 'python'. ~~python~~

pink

$$a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

→ gives the characters from given range.
→ returns true if a character exists in the

```
g = 'Hello'  
print('g is %s' % g) → True
```

not in → strong true if a character
does not exist in given str.
point ('d not in a) → True.

* Built-in `Stack` methods —

1) capitalize() → $a \leftarrow \text{ansay'}$
point(a.capitalize()) → this

Retaining a style where the 'i' character is upper case

Syntaxis → *Stung. Capital, 20.*

equation of motion.

~~function~~ Syntax → ~~str.~~ center(~~length, character~~)

Ps → Tf



- * A str is considered a valid identifier if it contains alphanumeric letters (A-Z, a-z, 0-9) or _.
- * A valid identifier cannot start with a non alphanumeric character.

3) casefold() :-
returns a str where all the characters are lowercase.

* This method is stronger, more aggressive, meaning that it will convert more characters to lowercase & will find more matches when comparing 2 strings so both are converted using casefold method.

4) strip() and lstrip() :-

Returns True if all characters are alphanumeric → `(A-Z, a-z, 0-9)`

spie characters X.

String → str.lstrip()

5) lalpha() →

Returns True if all characters are alphabetic (A-Z, a-z)

String → str.lalpha()

6) isdecimal() →

String → str.isdecimal()

7) isdigit() → Returns True if all characters are digits otherwise False.

String → str.isdigit()

8) lidentifer → returns true if str is a valid identifier, otherwise F.

① append() :-

String → list.append(obj).

Appends a list object into the existing list.

obj → object to be appended in the list.

[""] → list



tuple



tuple

eg → L = ["Ak programz", "Ak coding"]
L.append("Ak building")
print(L)

② insert() :- append() only works for

addition of elements at the end of list.
for addition of element at desired position, insert() is used.

Unlike append() which takes only argument, insert() requires 2 args (pos, value) pos=position.

Rntr → list.insert(pos, value)

pos → new specifying in which position to insert the value.

value → designated arg & can be

string or any type.(str, num.)

Q → L = [1, 2, 3, 4]
print("Akka", L)

L.insert(3, 2)

print(L)

→ 1, 2, 3, 2, 4
→ 1, 2, 3, 4, 8, 12

★ Basic tuple operations :-

1) length :-

eg → th = ("apple", "orange", "cherry")
print(len(th)) → 3

2) concatenation :-

You can use '+' operator to combine

2 tuples → .(.) .

eg → print((1, 2, 3) + (4, 5, 6)) → 1, 2, 3, 4, 5, 6

3) Repetition :-

repeat operator (*) for repeating the elements of a given num of times using the * operator.

eg → print("Ak") * 3 → AkAkAk

eg = L = [1, 2, 3, 'Ak', 'a', 'kk', 'c', 'Ak']
print(L.count('Ak')) → 3

2) Deleting a tuple :-

The elements of a tuple cannot be changed once it has been assigned. That also means you cannot remove/delete an item from a tuple.

But deleting a tuple entirely if possible using the keyword "del".

del "del"

eg = Ak = ("programz", "bk")
~~del(Ak)~~,
del(Ak).

5) Count() :- returning non 0 times a specified value occurs in a tuple.

Extr → tuple.count(value)

eg → t = (1, 3, 7, 7)

print(t, count(7)) → 2

6) Index() :- binds the 1st occurrence of the and specified values.

If there are exception of the value is not found.

Extr → tuple.index(value)

eg → t = (1, 3, 7, 8, 7, 5)

print(t, index(8)) → 3

⇒ Python dict =

* A dict is a collection of ordered unchanged & unorderd key:value pairs.

* In a object, keys must be unique & stored in an unorderd manner.

* eg → Ak = {'ab': 15, 'cd': 20}.

Point(Ak)

* Accessing elements from dict →

{name: 'John'
dict = {'name': 'John', 'age': 20, 'class': 'BSC CG'}
print(dict.get(name)) → John

* Indexing is used with other containing type to access values, dict uses keys.

* Key can be used either inside [] or .keys() the get method.

* Indexing → a = {'name': 'Ak', 'class': 'BSC CS'}

print(a['name']) → Ak.

* True diff - while using get() is treat it returning none instead of key error if the key is not found.

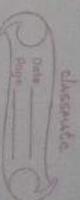
⇒ Dict on remove elements from dict :-

eg → Ak = {1:1, 2:4, 3:4, 4:2}
print(pop(3))

① pop()
② key/pattern
③ key/value:
④ clear()
on dict



classmate



classmate

This method removes an item with the provided key & returning the value.

- * `pop item()` used to remove & return an arbitrary item (key:value) from the dict.

All the items can be removed at once using the `clear()`.

You can also use the 'del' keyword to remove individual items or the entire dict itself.

eg → `s = {1:1, 2:4, 3:9, 4:16, 5:25}`

`print(s.pop(4))` → `{1:1, 2:4, 3:9, 5:25}`

`pprint(s.popitem())` → `{2:4, 3:9, 5:25}`

`del s[2]` → `{3:9, 5:25}`

`del s` → `{}.`

→ Built-in =

1) `all()` → Returns true if all keys of dict are true (empty dict is empty)

2) `any()` → Returning T if any key & dict is T, if the dict is empty return F.

3) `len()` → returning the length.

4) `cmp()` → compare items of 2 dict.

5) `sorted()` → return a new sorted list of keys in dict.

eg → `Ak = {1:1, 2:4, 3:9}` → 3

→ dict methods =

a) `clear()` = `dict.clear()`.

b) `copy()` = copy of the dict.

c) `get()` = returning value of specified key, dict.get(key,value).

→ Sets = `{ }.`

* A set is an unordered & unindexed collection

of items. (no relabel & placing)

* Every element is unique (no duplicates)

& must be immutable.

However the set itself is mutable you can add / remove items from it.

* A set is created placing all elements inside `{ }.`

* Sets can have any non of elements & they may be of any datatype.

eg → `set = {1, 2, 3}.`

`print(set)`.

* `set()` → A set can be passed to set()

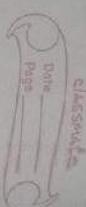
as well.

eg → # set obj char from str.

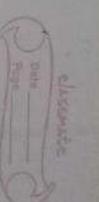
`AK = set('Ak programming')`

`print(AK)`.

* `set(str)` generates a set of chars in str.



union → |
intersection → &
difference → -



→ Accessing & changing set =

- * add() = you can add a single element

- * update() = you can add multiple elements!

It can take tuples, lists, & other sets as its arguments.

e.g. →

set = {1, 3}

print(set)

add element → set.add(2)

print(set)

add multiple element → set.update([2, 3, 4])

print(set)

→ {1, 3, 2, 4}

→ Removing elements from set =

- * A particular item can be removed from

Set using discard() & remove()

- * The only diff b/w 2 is that while

using discard(), if the the item does not exist in the set, it remains unchanged but remove() will raise an error in such condition.

e.g. →

set = {1, 3, 4, 5, 6}

print(set.discard(4))

print(set.remove(6))

→ {1, 3, 5}

→ Set operations =

- 1) Set union → union of A & B is a set of all elements from both sets.

- In Python, union is performed using |.

- same can be accomplished using union().

→ e.g. → A = {1, 2, 3, 4, 5}

B = {5, 6, 7}

print(A|B) → {1, 2, 3, 4, 5, 6, 7}

- using print(A.union(B)) → *

- 2) Set intersection → intersection of A & B is a set

- of elements that are common in both sets.

- It is performed using &.

- e.g. → print(A&B) → {5}.

- same can be accomplished by using intersection().

print(A.intersection(B)).

- 3) Set difference → Diff of A & B (A-B) is a set of elements that are only in A

- but not in B, similarly (B-A) set of elements in B but not in A.

- It is performed using - operator /

- difference() → {1, 2, 3, 4}

- e.g. → print(A-B) → {1, 2, 3, 4}

Q4 : functions

→ get factorial :-

- 1) all()
- 2) any()
- 3) len()
- 4) max()
- 5) min()
- 6) sum()

* () →

- * () is an organised, reusable, self-contained program segment that carries out some specific, well-defined task.
- * () contains its isolated action whenever it is accessed.

* 3 types of () →

- a) python built-in ()
- b) user defined ()
- c) anonymous () → no () name

* () elements → - name.

- parameters → input needed
return value

- return type | result type.

eg → def msg():
 print("Hello ~~World~~")

* parameter → create () →

eg → def msg(name):
 print("Hello " + name)
msg("A")

→ ()s →

a) abs() → absolute value of a non-negative
any may be an int / float.
Printer → abs(~~w~~).

eg → int = -20

eg → print(abs(int)) → 20//



3) all() → returns T if all items in an iterable are T, otherwise F.

If iterable obj is empty → returning T.
Syntax → all(iterable)
Print value → iterable ~~as~~ list.

eg → L = [0, 1, 2]
print(all(myList)) → T.

2) any() → anyone → T.

returning T, if any item in our iterable are T, otherwise F.

If use iterable obj is empty → F.

Syntax → any(iterable)
eg → L = [1, 3, 4, 0]
print(any(L)) → T.

L = [0, false]
print(any(L)) → F

3) ascii() → returns a unicode version of

a obj (str, tuples, list, etc).
This replace any non ascii char with escape char.

eg → \x00 → \u0000

R1, → ascii(obj)

eg → ca(a)

print(ascii(a,c)) → b5

4) bin() → returning binary version of a specified int.

Result will always start with prefix ob
5/2 → bin(5)

eg → Ak = 54
print(bin(Ak)) → 0b11010
(output shows start with)

5) bool() → returning boolean value of an specified obj.

obj will always return False unless

object is empty like [], (), {}
else {
① obj is False.
② obj is None.
③ obj is 0.
④ obj is empty.
⑤ obj is None.
else → True (obj)}

eg → to = []
print(bool(to)) → F.

6) chr() → returning char that represent the specified unicode.

8/x → chr(unicode).

number → int representing a

valid unicode point

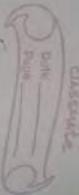
eg → print(chr(77)) → a.

7) complex() → returning a complex no. by specifying a real no. &

img no...

8/x → complex(real, img)

eg → print(complex(2, -3))



classmate

8) dict() → creates a dict & any
↳ key. dict(key word, arg)

9) float()

10) input()

11) hex()

12) int()

13) len() → no. of items in an obj.

14) list() → new list obj

15) max() → returning highest value.

16) min() → returning lowest value.

17) pow() → power(arg1, arg2)

18) print() → print(x,y,z)

19) round()

20) range() → range all around values.

21) set() → new set obj

22) sorted() → sorted(number [, args])

23) tuple() → new tuple

1) ord() → unicode char.
↳ char → ord('char').
eg → print(ord('5')) → 53.

2) print() → print(x,y,z).

3) pow(x,y,z) → optional.
eg → print(pow(4,2)) → 16

4) range()

5) round() → round(number [, args]).

6) set() → new set obj.
number → required, the no. to be rounded.

7) sorted() → sorted to use when rounding the no., relevant is 0.

8) tuple() → tuple(iterable, key=key, reversed=reversed)

9) dict() → dict(key word, arg). The sequence t.

10) next() → next(A)

11) ord() → ord('apple', 'banana')

12) print() → print(x)

13) print(A) → apple

14) print(min(A)) → banana

15) print(next(A)) → apple

16) print(sorted([1,5,3,0])) → [0,1,3,5]



classmate

17) ord() → unicode char.
↳ char → ord('char').
eg → print(ord('5')) → 53.

18) print() → print(x,y,z).

19) pow(x,y,z) → optional.
eg → print(pow(4,2)) → 16

20) print(sorted([1,5,3,0])) → [0,1,3,5]

21) print(next(A)) → apple

22) print(min(A)) → banana

23) print(sorted([1,5,3,0])) → [0,1,3,5]

24) print(tuple([1,2,3,4])) → (1,2,3,4)

25) print(set([1,2,3,4])) → {1,2,3,4}

26) print(dict([('apple', 'banana')])) → {'apple': 'banana'}

27) print(hex(10)) → '0xa'

28) print(int('0xa')) → 10

29) print(float('10.7')) → 10.7

30) print(str(10)) → '10'

31) print(bool(1)) → True

32) print(bool(0)) → False

33) print(bool('')) → False

34) print(bool([])) → False

35) print(bool({})) → False

36) print(bool(None)) → False

37) print(bool(False)) → False

38) print(bool(True)) → True

39) print(bool(None)) → False

40) print(bool(False)) → False

41) print(bool(True)) → True

42) print(bool(None)) → False

43) print(bool(False)) → False

44) print(bool(True)) → True

45) print(bool(None)) → False

46) print(bool(False)) → False

47) print(bool(True)) → True

48) print(bool(None)) → False

49) print(bool(False)) → False

50) print(bool(True)) → True

51) print(bool(None)) → False

52) print(bool(False)) → False

53) print(bool(True)) → True

54) print(bool(None)) → False

55) print(bool(False)) → False

56) print(bool(True)) → True

57) print(bool(None)) → False

58) print(bool(False)) → False

59) print(bool(True)) → True

60) print(bool(None)) → False

61) print(bool(False)) → False

62) print(bool(True)) → True

63) print(bool(None)) → False

64) print(bool(False)) → False

65) print(bool(True)) → True

66) print(bool(None)) → False

67) print(bool(False)) → False

68) print(bool(True)) → True

69) print(bool(None)) → False

70) print(bool(False)) → False

71) print(bool(True)) → True

72) print(bool(None)) → False

73) print(bool(False)) → False

74) print(bool(True)) → True

75) print(bool(None)) → False

76) print(bool(False)) → False

77) print(bool(True)) → True

78) print(bool(None)) → False

79) print(bool(False)) → False

80) print(bool(True)) → True

81) print(bool(None)) → False

82) print(bool(False)) → False

83) print(bool(True)) → True

84) print(bool(None)) → False

85) print(bool(False)) → False

86) print(bool(True)) → True

87) print(bool(None)) → False

88) print(bool(False)) → False

89) print(bool(True)) → True

90) print(bool(None)) → False

91) print(bool(False)) → False

92) print(bool(True)) → True

93) print(bool(None)) → False

94) print(bool(False)) → False

95) print(bool(True)) → True

96) print(bool(None)) → False

97) print(bool(False)) → False

98) print(bool(True)) → True

99) print(bool(None)) → False

100) print(bool(False)) → False

101) print(bool(True)) → True

102) print(bool(None)) → False

103) print(bool(False)) → False

104) print(bool(True)) → True

105) print(bool(None)) → False

106) print(bool(False)) → False

107) print(bool(True)) → True

108) print(bool(None)) → False

109) print(bool(False)) → False

110) print(bool(True)) → True

111) print(bool(None)) → False

112) print(bool(False)) → False

113) print(bool(True)) → True

114) print(bool(None)) → False

115) print(bool(False)) → False

116) print(bool(True)) → True

117) print(bool(None)) → False

118) print(bool(False)) → False

119) print(bool(True)) → True

120) print(bool(None)) → False

121) print(bool(False)) → False

122) print(bool(True)) → True

123) print(bool(None)) → False

124) print(bool(False)) → False

125) print(bool(True)) → True

126) print(bool(None)) → False

127) print(bool(False)) → False

128) print(bool(True)) → True

129) print(bool(None)) → False

130) print(bool(False)) → False

131) print(bool(True)) → True

132) print(bool(None)) → False

133) print(bool(False)) → False

134) print(bool(True)) → True

135) print(bool(None)) → False

136) print(bool(False)) → False

137) print(bool(True)) → True

138) print(bool(None)) → False

139) print(bool(False)) → False

140) print(bool(True)) → True

141) print(bool(None)) → False

142) print(bool(False)) → False

143) print(bool(True)) → True

144) print(bool(None)) → False

145) print(bool(False)) → False

146) print(bool(True)) → True

147) print(bool(None)) → False

148) print(bool(False)) → False

149) print(bool(True)) → True

150) print(bool(None)) → False

151) print(bool(False)) → False

152) print(bool(True)) → True

153) print(bool(None)) → False

154) print(bool(False)) → False

155) print(bool(True)) → True

156) print(bool(None)) → False

157) print(bool(False)) → False

158) print(bool(True)) → True

159) print(bool(None)) → False

160) print(bool(False)) → False

161) print(bool(True)) → True

162) print(bool(None)) → False

163) print(bool(False)) → False

164) print(bool(True)) → True

165) print(bool(None)) → False

166) print(bool(False)) → False

167) print(bool(True)) → True

168) print(bool(None)) → False

169) print(bool(False)) → False

170) print(bool(True)) → True

171) print(bool(None)) → False

172) print(bool(False)) → False

173) print(bool(True)) → True

174) print(bool(None)) → False

175) print(bool(False)) → False

176) print(bool(True)) → True

177) print(bool(None)) → False

178) print(bool(False)) → False

179) print(bool(True)) → True

180) print(bool(None)) → False

181) print(bool(False)) → False

182) print(bool(True)) → True

183) print(bool(None)) → False

184) print(bool(False)) → False

185) print(bool(True)) → True

186) print(bool(None)) → False

187) print(bool(False)) → False

188) print(bool(True)) → True

classmate
Date _____
Page _____

classmate
Date _____
Page _____

25) `str()` → convert to string
25) `sum()` → sum(`iterable, start`).
→ optional

26) `tuple()`
27) `types()`

```
from math import sqrt, log
```

→ `math()`

1) `math.ceil(x)` → for rounding.
2) `math.comb(n, r)` → for combination.

3) `" . factorial(x)`

4) `" . gcd(a, b)` → like LCM.

5) `" . sqrt(b)`

6) `" . remainder(x, y)`

7) `" . power(x, y)`

* `" . log10(x)`

8) `math.sin(r)`

* `" . sinh(x)`

9) `math.sinh(x)`

* `" . math.e`

10) `" . math.pi()`

11) `" . math.tan()`

→ `Date()` =

(2) `date.today()` → print local date
(3) `datetime.date(yr, month, day)`

```
[import datetime]
```

→ global variables → out.
local u → import a()

=> anonymous () = it is a () that is
defined without a name that uses
defined using lambda keyword
lambda → lambda arguments : exp.
log → double = lambda x : x * 2
print(double(5)) → 5 * 2 = 10