

## PYTHON NOTES

- |                                      |                      |
|--------------------------------------|----------------------|
| 1. Introduction                      | 2. Print function    |
| 3. Wearable declaration              | 4. Data types        |
| 5. Scientific notation of float data | 6. Operators         |
| 7. Conditional statements            | 8. loops             |
| 9. Control Float Statements          | 10. Type casting     |
| 11. Mutable and immutable            | 12. Object interning |
| 13. String                           | 14. List             |
| 15. Tuple                            | 16. Set              |
| 17. Dict                             | 18. Concatenation    |
| 19. sort and reverse                 | 20. Comprehension    |

## Introduction:

Python is a popular programming language. It was created by **Guido van Rossum**, and released in **1991**.

### Use Cases:

- Data Science, Machine Learning, GenAI.
- Web development (server-side).
- Software development.
- Mathematics.
- System scripting.

## Print Function:

`print("Sureshvj")` → Print a string

`print(x)` → Print a variable x

`print(x,y)` → Print two variables.

`print("Suresh, {0}, {1}".format(x,y))` → Print format str **way-1**

`print("Suresh, {}, {}".format(x,y))` → Print format str **way-2**

`print(f"Suresh {x}")` → Print format str **way-3**

`print("Python", end='@')` → end concatenates 2 print function messages with end value.

`print('09','12','2016', sep='-')` → sep will separate different values with sep value.

## Variable declaration:

`x = 10`

Declare single int variable

`x = "Suresh VJ"`

Declare single str variable

`x, y = 26, "Suresh VJ"`

Different memory location

`x, y = 5, 5`

Same memory location

Declare multiple variables

`x = y = "Suresh VJ"`

Same memory location

Declare multiple variables with single value

### Variable Declaration Rules:

- Variable name should not start with **num, special char, capital letter**. (1a, @x, Age)
- Variable name shouldn't contain the **spaces**. (sur name = 'vj')
- Variable name can start with **underscore**. (\_)

### Constant Variables:

- It is a special type of variable whose value should not change. Declared with capital letters.
- The constant variables declared in a separate python file (constant.py) and use those variables in another file (main.py) by importing them.

#### constant.py

```
# Declare constants
PI = 3.14
GRAVITY = 9.8
```

#### main.py

```
# Import constant file we created above
import constant

print(constant.PI) # prints 3.14
print(constant.GRAVITY) # prints 9.8
```

## Data Types:

Numeric data types	int, float, complex	26, 10.5, 2+3j
String data types	str	'Suresh VJ'
Sequence types	list, tuple, range	[], (), range(0,10)
Mapping data type	dict	{'key': value}
Set data types	set	{}
Boolean type	bool	True / False, 1 / 0
Null values	None	None

### Imp points:

- All data types are **objects**.
- All data types have **immutable** property except list, set, dict.
- All data types have **object intern** properties.

### Some data which support by python:

Long int	9618112600	-----
Binary	0b0110101	0b-----
Decimal	100	100---
Octal	0o215	0o----
Hexa-decimal	0x12d	0x---d

## Scientific Notation of Float Data:

```
x = 35e3      # 35000.0
y = 12E4      # 120000.0
z = -87.7e100 # -8.77e+101
print(x, y, z)
```

**Precision:** Float has a fixed number of bits, leading to precision limits.

**Scientific Notation:** Allows expressing large or small numbers compactly using **e or E** to denote powers of 10.

### Examples:

- **35e3** means  $35 \times 10^3$  which is **35000**.
- **12E4** means  $12 \times 10^4$  which is **120000**.
- **-87.7e100** means  $-87.7 \times 10^{100}$ .

## Operators:

Arithmetic operators	+, -, /, //, %, *, **
Comparison operators	<, >, <=, >=, ==, !=
Assignment operators	=, +=, -=, /=, //=, %=, *=, **=
Logical operators	and, or, not
Identical operator	is, in (is not, not in)

## Conditional Statements:

### if:

```
if condition:  
    # code
```

### Nested if:

```
if condition_1:  
    # code  
    if condition_2:  
        # code
```

### if - else:

```
if condition:  
    # code  
else:  
    #code
```

### if - else:

```
if condition:  
    if condition:  
        # code  
else:  
    if condition:  
        # code
```

### elif:

```
if condition_1:  
    # code  
elif condition_2:  
    #code  
elif condition_3:  
    #code  
else:  
    #code
```

### Advanced Syntax:

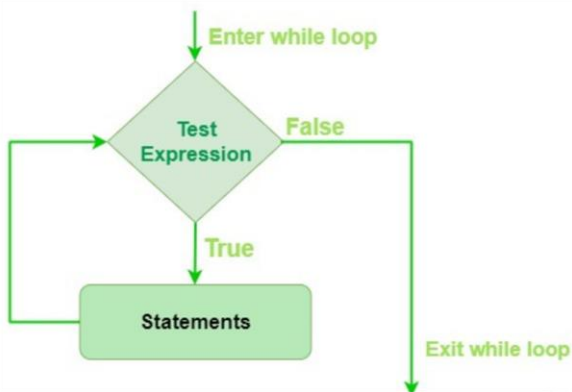
```
print("VJ") if <condition> else print("R")  
print("A") if <condition_1> else print("B") if <condition_2> else print("C")
```

### Imp Points:

`elif` is also possible without `else`.

## loops:

### While



```
while <condition>:
    # Code
```

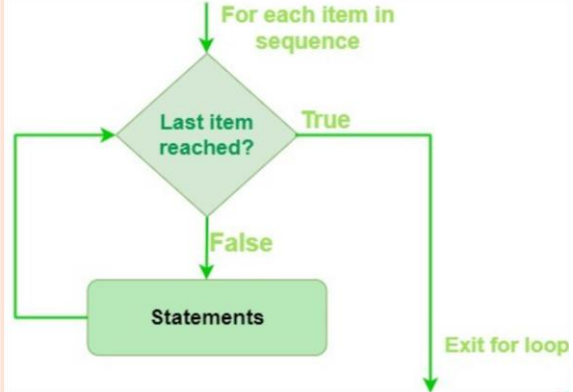
### While with else

```
while <condition>:
    # Code
else:
    #code
```

### Infinity while

```
while True:
    # Code
```

### For:



```
for i in <iterator>:
    # code
```

### For with else

```
for i in range(6):
    #code
else:
    print("Finally finished!")
```

### For - else with break

```
for x in range(6):
    if x == 3: break
    print(x)
else:
    print("Finally finished!")
# The else not execute if loop
breaks
```

## Control Flow Statements:

### Pass

Pass Does nothing, just a placeholder.

```
for i in range(5):
    if i == 3:
        pass
    else:
        print(i)
```

# Do nothing when i equals 3

### Break:

Break exits the loop immediately.

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

# Exit the loop when i equals 3

### Continue:

Continue skips the rest of the loop and starts the next iteration.

```
for i in range(5):
    if i == 3:
        continue
    print("X")
    print(i)
```

# Skip printing when i equals 3

## Type Casting:

The below constructors are used to perform the type casting.

int()	float()	complex()
bool()	str()	list()
tuple()	set()	dict()

from	int	float	complex	bool	str	list	tuple	set	dict
int	✓	✓	✓	1/0 CK	✓	X	X	X	X
float	✓	✓	✓	CK	✓	X	X	X	X
bool	✓	✓	✓	T/F	✓	X	X	X	X
complex	X	X	✓	CK	✓	X	X	X	X
str	✓	✓	X	CK	✓	✓	✓	✓	X
list	X	X	X	CK	✓	✓	✓	✓	X
tuple	X	X	X	CK	✓	✓	✓	✓	X
set	X	X	X	CK	✓	✓	✓	✓	X
dict	X	X	X	CK	✓	keys	keys	keys	✓

## Mutable & Immutable:

### Mutable:

If data can be changeable or updatable in current memory location then that objects are called as mutable.

List Set Dict

### Immutable:

If data can't be changeable or updatable in current memory location then that objects are called as immutable.

Int Float Bool Str Tuple None

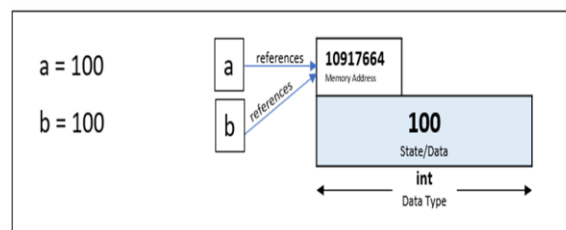
## Obj interning:

Object Interning is nothing but the two different variables having the same value is stored in the same address

If two variables / objects having same data, Python creates only one object and save that data in one instance only and provide the object address to both variables.

Eligible to interning property:

Int Float Bool Complex Str



Eligible to interning property:

List Tuple Set Dict

**String:**

Properties:

Interned obj

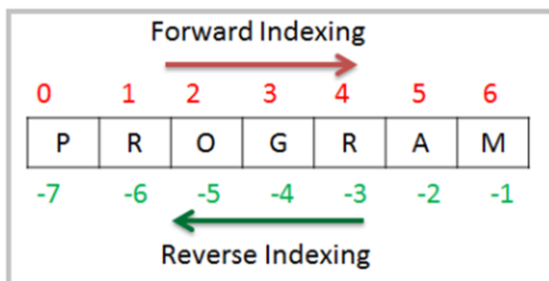
- String index numbers starts from 0 in forward direction, and -1 in reverse direction.

Syntax	Explanation
s.capitalize()	Capitalize the starting character of the string and rest of all characters will be converted into lower case.
s.title()	title the starting character of each word in a string and rest of all characters will be converted into lower case.
s.casefold()	Used to convert string to lower case. It is similar to lower() string method, but case <b>removes all the case distinctions</b> present in a string. ("Groß")
s.lower()	Used for converting into lowercase
s.upper()	Used for converting into uppercase
s.swapcase()	Converts all uppercase characters to lowercase and vice versa
s.istitle()	It returns True if all the words in the string are title cased, otherwise returns False.
s.islower()	It returns <b>True</b> if all alphabets in a string are in lowercase. otherwise returns <b>False</b> .
s.isupper()	It returns <b>True</b> if all alphabets in a string are in uppercase. otherwise returns <b>False</b> .
s.center(4, '*') s.center(4)	It will return a new string which contains 4 * s before and after the input string "S".
s.strip() s.strip(s1)	It Remove spaces / specified characters from starting and ending of the string.
s.rstrip() s.rstrip(s1)	It Remove spaces / specified characters from right side of the string.
s.lstrip() s.lstrip(s1)	It Remove spaces / specified characters from left side of the string.
s.count('sub_str')	Returns the number of occurrences of a substring in the given string
s.find('sub_str')	Returns the lowest index or first occurrence of the substring if it is found in a given string. If it is not found, then it returns -1.
s.rfind('sub_str')	Returns the rightmost index of the substring if found in the given string. If not found then it returns -1.

<code>s.startswith('sub_str')</code>	Returns <b>True</b> if a string starts with the specified prefix ('sub_str'), otherwise returns <b>False</b> .
<code>s.endswith('sub_str')</code>	Returns True if a string ends with the given suffix ('sub_str'), otherwise returns False.
<code>s.index('sub_str')</code>	Returns index of the first occurrence of an existing substring inside a given string. Otherwise, it raises <b>ValueError</b> .
<code>s.rindex('sub_str')</code>	Highest index of the substring inside the string if the substring is found. Otherwise, it raises <b>ValueError</b> .
<code>s.isnumeric()</code>	Returns <b>"True"</b> if all characters in the string are numeric characters, otherwise returns <b>"False"</b> .
<code>s.isalnum()</code>	It checks whether all the characters in a given string are either alphabet or numeric (alphanumeric) characters.
<code>s.isalpha()</code>	It is used to check whether all characters in the String is an alphabet.
<code>s.isdigit()</code>	Returns <b>"True"</b> if all characters in the string are digits, Otherwise, It returns <b>"False"</b> .
<code>s.isdecimal()</code>	Returns true if all characters in a string are decimal, else it returns False.
<code>s.isspace()</code>	Returns <b>"True"</b> if all characters in the <u>string</u> are whitespace characters, Otherwise, It returns <b>"False"</b> . This function is used to check if the argument contains all whitespace characters, such as: <ul style="list-style-type: none"> <li>• ' ' – Space</li> <li>• '\t' – Horizontal tab</li> <li>• '\n' – Newline</li> <li>• '\v' – Vertical tab</li> <li>• '\f' – Feed</li> <li>• '\r' – Carriage return</li> </ul>

### Slicing:

- In python, all sequence data index numbers start from 0 in Forward direction and start with -1 in reverse direction.
- Slicing is a process used to extract a subset of elements from a sequence.



In slicing the selection direction and step direction should be same. otherwise, it will not work.

**Syntax:** `variable[start:stop:step]`

**Ex:**

```
x = "PROGRAM"
x[3:7] # GRAM
```

**start:** The index at which the slice starts (inclusive). If omitted, it defaults to the beginning of the sequence.

**stop:** The index at which the slice ends (exclusive). If omitted, it defaults to the end of the sequence.

**step:** The interval between elements in the slice. If omitted, it defaults to 1.



<i>Start to Stop Direction</i>	<i>Step Direction</i>	<i>Example (sequence = [0, 1, 2, 3, 4, 5])</i>	<i>Result</i>	<i>Works</i>
Positive	Positive	sequence[1:4:1]	[1, 2, 3]	Yes
Positive	Negative	sequence[1:4:-1]	[]	No
Positive	Positive	sequence[-4:-1:1]	[2, 3, 4]	Yes
Negative	Negative	sequence[-1:-4:-1]	[5, 4, 3]	Yes
Positive	Positive	sequence[1:-1:1]	[1, 2, 3, 4]	Yes
Positive	Negative	sequence[1:-1:-1]	[]	No
Positive	Positive	sequence[-4:4:1]	[2, 3, 4]	Yes
Positive	Negative	sequence[-4:4:-1]	[]	No
Negative	Positive	sequence[-1:1:1]	[]	No
Negative	Negative	sequence[4:-4:-1]	[4, 3, 2, 1]	Yes

### List:

Declaration: `[]`, `list()`

Properties:

Mutable	Allow duplicates
Ordered	Not interned obj
Sliceable	Allow all data types
Non-inclusive	

- Declaration Possible ways:  
`[]`, `[4]`, `[4, ]`, `[4, ]`
- List index numbers starts from 0 in forward direction, and -1 in reverse direction.

<code>l.append(val)</code>	Append the value end of the list
<code>l.extend([val, val, ...])</code>	Add provided list of values at end
<code>l.insert(idx, val)</code>	Insert a value at a particular index position
<code>l.copy()</code>	Copy the list into another variable.
<code>l.count(val)</code>	Returns the frequency of a value from a list.
<code>l.index(val)</code>	Return the index number of a value.
<code>l.reverse()</code>	Reverse the list.
<code>l.sort(reverse= T / F)</code>	Sort the list – default ascending order (reverse= False)
<code>l.pop(idx)</code>	Remove specified indexed value - default remove last value
<code>l.remove(val)</code>	Remove first occurrence of the specified value
<code>l.clear()</code>	Clear the list object from memory

## Tuple:

Declaration: `()`, `tuple()`

Properties:

Immutable	Allow duplicates
Ordered	Not interned obj
Sliceable	Allow all data types
Non-inclusive	

- Declaration Possible ways:

```
t = 1,2,3  
( ), 4,, (4, )
```

- Tuple index numbers starts from 0 in forward direction, and -1 in reverse direction.

```
t.count(val)
```

Returns the frequency of a value from a list.

```
t.index(val)
```

Return the index number of a value.

## Set:

Declaration: `{}`, `set()`

Properties:

Mutable	Not allow duplicates
Not ordered	Not interned obj
Can't sliceable	Not allow dict, list, set

- Declaration Possible ways:

```
{}, {4}, {4,}, {4, }
```

- Set allows only mutable data types.

```
s.add(val)
```

Add a value to set

```
s.clear()
```

Remove all values from set

```
s.copy()
```

Return a copy of the set

```
s1.difference(s2)
```

Returns difference (*items exist only in the first set*) between two sets.

```
s1.difference_update(s2)
```

Update the set s1 with items which are not existed in s2.

```
s.discard("val")
```

Remove a specified item

```
s1.intersection(s2)
```

Returns a set with items which are present in both s1, s2 sets.

```
s1.intersection_update(s2)
```

Removes the items from s1 which are not present in s2.

```
l.pop(idx)
```

Remove specified indexed value - default remove last value

```
l.remove(val)
```

Remove first occurrence of the specified value

```
l.clear()
```

Clear the list object from memory

## Dict:

Declaration: `{'key': 'value'}`, `dict()`

Properties:

- Mutable
- Ordered(from 3.7)
- Non-Sliceable
- Allow duplicate values, not keys
- Not interned obj
- Key should be immutable,
- value can be anything

- Declaration Possible ways:

```
{'key_1': 'value_1', 'key_2': 'value_2'}
```

```
dict(key_1='value_1', key_2='value_2')
```

Method	Description
<code>d.clear()</code>	Removes all items from the dictionary.
<code>d.copy()</code>	Returns a shallow copy of the dictionary.
<code>d.fromkeys(seq[, v])</code>	Creates a new dictionary with keys from <code>seq</code> and values set to <code>v</code> (default is <code>None</code> ).
<code>d.get(key[, default])</code>	Returns the value for <code>key</code> if <code>key</code> is in the dictionary, else <code>default</code> .
<code>d.items()</code>	Returns a view object with a list of dictionary's key-value tuple pairs.
<code>d.keys()</code>	Returns a view object with a list of all the keys in the dictionary.
<code>d.pop(key[, default])</code>	Removes the specified key and returns the corresponding value. If key is not found, <code>default</code> is returned if provided, otherwise <code>KeyError</code> is raised.
<code>d.popitem()</code>	Removes and returns a (key, value) pair from the dictionary. Pairs are returned in LIFO (last-in, first-out) order in Python 3.7+.
<code>d.setdefault(key[, default])</code>	Returns the value of <code>key</code> if <code>key</code> is in the dictionary, else inserts <code>key</code> with a value of <code>default</code> and returns <code>default</code> .
<code>d.update([other])</code>	Updates the dictionary with the key-value pairs from <code>other</code> , overwriting existing keys.
<code>d.values()</code>	Returns a view object with a list of all the values in the dictionary.
<code>d.__contains__(key)</code>	Checks if the dictionary contains the specified key (e.g., <code>key in d</code> ).
<code>d.__delitem__(key)</code>	Deletes the specified key from the dictionary (e.g., <code>del d[key]</code> ).
<code>d.__getitem__(key)</code>	Returns the value associated with the specified key (e.g., <code>d[key]</code> ).
<code>d.__setitem__(key, value)</code>	Sets the value associated with the specified key (e.g., <code>d[key] = value</code> ).
<code>d.__len__()</code>	Returns the number of items in the dictionary (e.g., <code>len(d)</code> ).

### Concatenation:

Concatenation is the process of extend the value with new value.

**Ex:** `a = "Suresh", b = " VJ"`  
`b` concatenates with `a` is `"Suresh VJ"`

1. str with str concatenation is possible.
2. list with list concatenation is possible.
3. tuple with tuple concatenation is possible.

`a+b, a+=b`

we can do concatenation by above ways

### Sort & Reverse:

#### Sort:

`sortend(x) → ascending`

`sortend(x, reverse=True) → descending`

- When we sort the string, that returns list of characters. If we want to convert that list into str then use `"".join(output_list)`
- List has by its own sort function `1.sort()`
- Sort applicable to **Str List Tuple Set Dict**

#### Reverse:

`x[::-1]`

`Reversed(x)`

- Can't apply reverse operation on **Set** and **Dict**

### Comprehension:

Let's consider list x as below & applying comprehension in 3 way i.e *with out condition*, *with if*, *with if else*

```
x = range(0,11)
```

```
lst = [i+2 for i in x]
```

```
lst = [i+2 for i in x if i <= 5]
```

```
lst = [i+2 if i > 3 else i for i in x]
```

This concept applicable to:

**List Tuple Set Dict**

For **Dict** we should pass key value pair as below:

```
lst = {"key{i}" : i+2 for i in x}
```