print("Sureshvj") 🡪 Print a string

print(x) 🡪 Print a variable x

print(x,y) 🡪 Print two variables.

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

print(f"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.

***Print Function:***

***Variable declaration:***

|  |  |
| --- | --- |
| x = 10 | Declare single int values |
| x = "Suresh VJ" | Declare single str values |
| x, y = 26, "Suresh VJ" | Declare multiple values |

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.** ( \_ )

|  |  |  |
| --- | --- | --- |
| 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 | 9618112600L | -----L | | Binary | 0b0110101 | 0b----- | | Decimal | 100 | 100--- | | Octal | 0c215 | 0c---- | | Hexa-decimal | 0x12d | 0x---d | |

***Data Types:***

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

***Operators:***

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 | ✓ | |

***Type Casting:***

***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. | Optimization in Python — Interning | by Chetan Ambi | Towards Data Science |
| *Eligible to interning property:*  Int Float Bool Complex Str | *Eligible to interning property:*  List Tuple Set Dict |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Declaration: '', "", ''' ''', """ """  Properties:   |  |  | | --- | --- | | Immutable | Interned obj | | Ordered |  | | Sliceable |  | | Non-inclusive |  | | * String index numbers starts from 0 in forward direction, and -1 in reverse direction. |

***String:***

|  |  |
| --- | --- |
| ***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**removes all the case distinctions** present in a string. |
| 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 **True**if all alphabets in a string are in lowercase. otherwise returns **False**. |
| s.isupper() | It returns **True**if all alphabets in a string are in uppercase. otherwise returns **False**. |
| 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. |
| s.startswith('sub\_str') | Returns **True** if a string starts with the specified prefix ('sub\_str'), otherwise returns **False**. |
| s.endswith('sub\_str') | Returns True if a string ends with the given suffix ('sub\_str'), otherwise returns False. |
| s.index('sub\_str') | Returns index of the first occurrence of an existing substring inside a given string. Otherwise, it raises **ValueError**. |
| s.rindex('sub\_str') | Highest index of the substring inside the string if the substring is found. Otherwise, it raises **ValueError**. |
| s.isnumaric() | Returns “**True**” if all characters in the string are numeric characters, otherwise returns “**False**”. |
| s.isalnum() | It checks whether all the characters in a given string are either alphabet or numeric (alphanumeric) characters. |
| s.isalpha() | It is used to check whether all characters in the String is an alphabet. |
| s.isdisit() | Returns “**True**” if all characters in the string are digits, Otherwise, It returns “False”. |
| s.isdecimal() | Returns true if all characters in a string are decimal, else it returns False. |
| s.isspace() | Returns “**True**” if all characters in the [string](https://www.geeksforgeeks.org/python-string/) are whitespace characters, Otherwise, It returns “**False**”. This function is used to check if the argument contains all whitespace characters, such as:   * ‘ ‘ – Space * ‘\t’ – Horizontal tab * ‘\n’ – Newline * ‘\v’ – Vertical tab * ‘\f’ – Feed * ‘\r’ – Carriage return |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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. |

|  |  |
| --- | --- |
| l.append(val) | Append the value end of the list |
| l.extend([val, val, ..]) | Add provided list of values at end |
| l.insert(idx, val) | Insert a value at a particular index position |
| l.copy() | Copy the list into another variable. |
| l.count(val) | Returns the frequency of a value from a list. |
| l.index(val) | Return the index number of a value. |
| l.reverse() | Reverse the list. |
| l.sort(reverse= T / F) | Sort the list – default ascending order (reverse= False) |
| 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 |

***List:***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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. |

***Tuple:***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

***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. |
| s1.intersection(s2) | Returns a set with items which are present in both s1, s2 sets. |
| l.reverse() | Reverse the list. |
| l.sort(reverse= T / F) | Sort the list – default ascending order (reverse= False) |
| 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 |

***Set:***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Slicing is the process of accessing a small chunk / subpart of a sequence data.  Syntax:  slice = X[start : end : step] | | |  |  |  |  | | --- | --- | --- | --- | | State | Index | Step | output | | Forword | +ve | +ve | Forword | | Reverse | +ve | -ve | Reverse | | Forword | -ve | +ve | Forword | | Reverse | -ve | -ve | Reverse | | |
| X[2:5:2] | Start from 2 and ends at 4 and step by 2 | | Note:   * Slicing excluded the end position index number. * We can slice Forword and reverse direction. |
| X[-2:-5:-1] | Start from -2 and ends at -4 step by -1 | |
| X[2:5] | Start from 2 and ends at 4 | |
| X[:5] | From 0 to 4 | |
| X[2:] | From 2 to end | |
| X[:] | Starting to ending | |

***Slicing:***

|  |  |
| --- | --- |
| 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 |

***Concatenation:***

|  |  |
| --- | --- |
| ***Sort:***  sortend(x) 🡪 ascending  sortend(x, reverse=True) 🡪 descending   * When we sort the string, that returns list of characters. If we want to converts that list into str then use "".join(output\_list) * List has by its own sort function l.sort() * Sort applicable to Str List Tuple Set Dict | ***Reverse:***  x[::-1]  Reversed(x)   * Can’t apply reverse operation on Set and Dict |

***Sort & Reverse:***

***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 = {f"key{i}" : i+2 for i in x} |