Contents

Variable

2myvar = "John" my-var = "John" my var = "John"	myvar = "John"
Output: SyntaxError: invalid syntax	**Output:** John

Number Type

Integer (int)	Float (float)	Complex (complex)
x = 1 y = 35656222554887711 z = -3255522 print(type(x)) print(type(y)) print(type(z))	x = 1.10 y = 1.0 z = -35.59 a = 35e3 b = 12E4 c = -87.7e100 print(type(x)) print(type(y)) print(type(z)) print(type(a)) print(type(b)) print(type(c))	x = 3+5j y = 5j z = -5j print(type(x)) print(type(y)) print(type(z))
<class 'int'> <class 'int'> <class 'int'>	<class 'float'> <class 'float'> <class 'float'> <class 'float'> <class 'float'> <class 'float'>	<class 'complex'> <class 'complex'> <class 'complex'>

Type Conversion

Code	Output

x = 1 # int
y = 2.8 # float
z = 1j # complex
#convert from int to float:
a = float(x)
#convert from float to int:
b = int(y)
#convert from int to complex:
c = complex(x)
print(a)
print(b)
print(c)
	1.0
	2
	(1+0j)
	<class 'float'>
	<class 'int'>
	<class 'complex'>

`print(type(a))`
`print(type(b))`
`print(type(c))`	

Type Casting

```
x = float(1) # x will be 1.0
y = int(2.8) # y will be 2
z = str(3.0) # z will be 3.0
```

Logical

And (&)		
A	**B**	**Result**
0	0	0
0	1	0
1	0	0
1	1	1
OR (|)		
A	**B**	**Result**
0	0	0
0	1	1
1	0	1
1	1	1
XOR (^)		
A	**B**	**Result**
0	0	0
0	1	1
1	0	1
1	1	0
NOT		
A		**Result**
1		0
0		1

Boolean

Code	**Output**
print(bool(False)) print(bool(None)) print(bool(0)) print(bool("")) print(bool(())) print(bool([])) print(bool({}))	False False False False False False False

Function to return Boolean

Code	Output

`def myFunction() :`
`  return True`
`print(myFunction())`
	`True`

Check if an object is an integer or not:

Code	Output
`x = 200` `print(isinstance(x, int))`	`True`

Operators and Associativity

Operator	Type	Associativity
( )	Parentheses	left-to-right
**	Exponent	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
is, is not in, not in	Identity Membership operators	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
not	Logical NOT	right-to-left
and	Logical AND	left-to-right
or	Logical OR	left-to-right
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left

Strings are Array

Code	Output
`a = "Hello, World!"` `print(a[1])`	e

for loop in string

Code	Output
`for x in "banana":` `  print(x)`	b a n a n a

String length

Code	Output
`a = "Hello, World!"` `print(len(a))`	13

in statement in string

Code	Output
`txt = "The best things in life are free!"` `print("free" in txt)`	True

if loop in string

Code	Output

`txt = "The best things in life are free!"`
`if "free" in txt:`
`  print("Yes, 'free' is present.")`
	Yes, 'free' is present.

if not in string

Code	Output
`txt = "The best things in life are free!"` `print("expensive" not in txt)`	True

if statement in string

Code	Output

txt = "The best things in life are free!" if "expensive" not in txt: print("No, 'expensive' is NOT present.")	No, 'expensive' is NOT present.

Get character from a position in string

Code	Output
b = "Hello, World!" print(b[2:5])	llo

Get character from start of a string

Code	Output
b = "Hello, World!" print(b[:5])	Hello

Get character to end if a string

Code	Output
b = "Hello, World!" print(b[2:])	llo, World!

Negative Indexing in string

Code	Output
b = "Hello, World!" print(b[-5:-2])	orl

upper method in string

Code	Output
a = "Hello, World!" print(a.upper())	HELLO, WORLD!

lower method in string

Code	Output
a = "Hello, World!" print(a.lower())	hello, world!

remove whitespace in string\strip method in string

Code	Output
a = " Hello, World! " print(a.strip()) # returns "Hello, World!"	Hello, World!

replace method in string

Code	Output
a = "Hello, World!" print(a.replace("H", "J"))	Jello, World!

Split string

Code	Output
a = "Hello, World!" print(a.split(",")) # returns ['Hello', ' World!']	['Hello', ' World!']

String Concatenation

Code	Output
a = "Hello" b = "World" c = a + b print(c)	HelloWorld

Adding space in between two strings

Code	Output
a = "Hello" b = "World" c = a + " " + b print(c)	Hello World

format method in string

Code	Output
age = 36 txt = "My name is John, and I am {}" print(txt.format(age))	My name is John, and I am 36

format method in string for multiple argument

Code	Output
quantity = 3 itemno = 567 price = 49.95 myorder = "I want {} pieces of item {} for {} dollars." print(myorder.format(quantity, itemno, price))	I want 3 pieces of item 567 for 49.95 dollars.

Index number in string

Code	Output
quantity = 3 itemno = 567 price = 49.95 myorder = "I want to pay {2} dollars for {0} pieces of item {1}." print(myorder.format(quantity, itemno, price))	I want to pay 49.95 dollars for 3 pieces of item 567.

Error in String

age = 36 txt = "My name is John, I am " + age print(txt)	age = "36" txt = "My name is John, I am " + age print(txt)
Output: TypeError: must be str, not int	**Output:** My name is John, I am 36

Capitalize method in string

Code	Output
txt = "hello, and welcome to my world." x = txt.capitalize() print (x)	Hello, and welcome to my world.

Casefold method in string

Code	Output
txt = "Hello, And Welcome To My World!" x = txt.casefold() print(x)	hello, and welcome to my world!

Center method in string

Code	Output
txt = "banana" x = txt.center(20) print(x)	banana

Count method in string

Code	Output
txt = "I love apples, apple are my favorite fruit"	2

`x = txt.count("apple")` `print(x)`	

encode method in string

Code	Output

`txt = "My name is Ståle"`
`x = txt.encode()`
`print(x)`
	`b'My name is St\xc3\xa5le'`

endswith method in string

Code	Output

`txt = "Hello, welcome to my world."`
`x = txt.endswith(".")`
`print(x)`
	`True`

expandtabs method in string

Code	Output

`txt = "H\te\tl\tl\to"`
`x =  txt.expandtabs(2)`
`print(x)`
	`H e l l o`

find method in string

Code	Output

`txt = "Hello, welcome to my world."`
`x = txt.find("welcome")`
`print(x)`
	`7`

isalnum method in string

Code	Output

`txt = "Company12"`
`x = txt.isalnum()`
`print(x)`
	`True`

isalpha method in string

Code	Output

`txt = "CompanyX"`
`x = txt.isalpha()`
`print(x)`
	`True`

isdecimal method in string

Code	Output

txt = "\u0033" #unicode for 3
x = txt.isdecimal()
print(x)
	True

isdigit method in string

Code	Output
txt = "50800" x = txt.isdigit() print(x)	True

isidentifier method in string

Code	Output
txt = "Demo" x = txt.isidentifier() print(x)	True

islower method in string

Code	Output
txt = "hello world!" x = txt.islower() print(x)	True

isnumeric method in string

Code	Output
txt = "565543" x = txt.isnumeric() print(x)	True

isprintable method in string

Code	Output
txt = "Hello! Are you #1?" x = txt.isprintable() print(x)	True

isspace method in string

Code	Output
txt = " " x = txt.isspace() print(x)	True

istitle method in string

Code	Output
txt = "Hello, And Welcome To My World!" x = txt.istitle() print(x)	True

isupper method in string

Code	Output
txt = "THIS IS NOW!" x = txt.isupper() print(x)	True

join method in string

Code	Output
myTuple = ("John", "Peter", "Vicky") x = "#".join(myTuple) print(x)	John#Peter#Vicky

ljust method in string

Code	Output
txt = "banana" x = txt.ljust(20) print(x, "is my favorite fruit.")	banana is my favorite fruit.

lstrip method in string

Code	Output
txt = " banana " x = txt.lstrip() print("of all fruits", x, "is my favorite")	of all fruits banana is my favorite

maketrans method in string

Code	Output
txt = "Hello Sam!" mytable = txt.maketrans("S", "P") print(txt.translate(mytable))	Hello Pam!

partition method in string

Code	Output

txt = "I could eat bananas all day" x = txt.partition("bananas") print(x)	('I could eat ', 'bananas', ' all day')

rfind method in string

Code	Output
txt = "Mi casa, su casa." x = txt.rfind("casa") print(x)	12

rindex method in string

Code	Output
txt = "Mi casa, su casa." x = txt.rindex("casa") print(x)	12

rjust method in string

Code	Output
txt = "banana" x = txt.rjust(20) print(x, "is my favorite fruit.")	banana is my favorite fruit.

rpartition method in string

Code	Output
txt = "I could eat bananas all day, bananas are my favorite fruit" x = txt.rpartition("bananas") print(x)	('I could eat bananas all day, ', 'bananas', ' are my favorite fruit')

rsplit method in string

Code	Output
txt = "apple, banana, cherry" x = txt.rsplit(", ") print(x)	['apple', 'banana', 'cherry']

rstrip method in string

Code	Output
txt = " banana " x = txt.rstrip() print("of all fruits", x, "is my favorite")	of all fruits banana is my favorite

splitlines method in string

Code	Output
txt = "Thank you for the music\nWelcome to the jungle" x = txt.splitlines() print(x)	['Thank you for the music', 'Welcome to the jungle']

startswith method in string

Code	Output
txt = "Hello, welcome to my world." x = txt.startswith("Hello") print(x)	True

swapcase method in string

Code	Output
txt = "Hello My Name Is PETER" x = txt.swapcase() print(x)	hELLO mY nAME iS peter

title method in string

Code	Output
txt = "Welcome to my world" x = txt.title() print(x)	Welcome To My World

translate method in string

Code	Output

#use a dictionary with ascii codes to replace 83 (S) with 80 (P):
mydict = {83: 80}
txt = "Hello Sam!"
print(txt.translate(mydict))
	Hello Pam!

upper method in string

Code	Output
txt = "Hello my friends" x = txt.upper() print(x)	HELLO MY FRIENDS

zfill method in string

Code	Output
txt = "50" x = txt.zfill(10) print(x)	0000000050

Escape Character

Code	Type	Code	Output
\'	Single Quote	txt = 'It\'s alright.' print(txt)	It's alright.
\\	Backslash	txt = "This will insert one \\ (backslash)." print(txt)	This will insert one \ (backslash).
\n	New Line	txt = "Hello\nWorld!" print(txt)	Hello World!
\r	Carriage Return	txt = "Hello\rWorld!" print(txt)	Hello World!
\t	Tab	txt = "Hello\tWorld!" print(txt)	Hello World!
\b	Backspace	txt = "Hello \bWorld!" print(txt)	HelloWorld!
\ooo	Octal Value	txt = "\110\145\154\154\157" print(txt)	Hello
\xhh	Hex Value	txt = "\x48\x65\x6c\x6c\x6f" print(txt)	Hello

txt = "We are the so-called "Vikings" from the north."	txt = "We are the so-called \"Vikings\" from the north."
Output: SyntaxError: invalid syntax	**Output:** We are the so-called "Vikings" from the north.

Creating List

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Code	Output

Length of List

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Code	Output

Data Type of List Item

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

Code	Output

Mixed Data Type List

```
list1 = ["abc", 34, True, 40, "male"]
```

Code	Output

Type method in list

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

Code	Output

List constructor in list

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

Code	Output

Accessing List

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Code	Output

Negative Index in List

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Code	Output

Range of Index in list

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Code	Output

Printing first range index in list

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Code	Output

Printing after a range index in list

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

Code	Output

Range of Negative Indexes in list

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Code	Output

Check if Item Exists\ Using in keyword in list

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Code	Output

Change a range of item values in list

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Code	Output

Change the second value by replacing it with two new values

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

Code	Output

Change the second and third value by replacing it with one value in list

```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

Code	Output

insert method in list

Code	Output

`thislist = ["apple", "banana", "cherry"]`
`thislist.insert(2, "watermelon")`
`print(thislist)`
	`['apple', 'banana', 'watermelon', 'cherry']`

append method in list

Code	Output

`thislist = ["apple", "banana", "cherry"]`
`thislist.append("orange")`
`print(thislist)`
	`['apple', 'banana', 'cherry', 'orange']`

insert method in list

Code	Output

`thislist = ["apple", "banana", "cherry"]`
`thislist.insert(1, "orange")`
`print(thislist)`
	`['apple', 'orange', 'banana', 'cherry']`

Add the elements of two list

Code	Output

`thislist = ["apple", "banana", "cherry"]`
`tropical = ["mango", "pineapple", "papaya"]`
`thislist.extend(tropical)`
`print(thislist)`
	`['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']`

Add elements of a tuple to a list

Code	Output

`thislist = ["apple", "banana", "cherry"]`
`thistuple = ("kiwi", "orange")`
`thislist.extend(thistuple)`
`print(thislist)`
	`['apple', 'banana', 'cherry', 'kiwi', 'orange']`

remove method in list

Code	Output
`thislist = ["apple", "banana", "cherry"]`	`['apple', 'cherry']`

thislist.remove("banana") print(thislist)	

pop method in list

Code	Output
thislist = ["apple", "banana", "cherry"] thislist.pop(1) print(thislist)	['apple', 'cherry']

remove the list item from list

Code	Output
thislist = ["apple", "banana", "cherry"] thislist.pop() print(thislist)	['apple', 'banana']

Remove first item in list

Code	Output
thislist = ["apple", "banana", "cherry"] del thislist[0] print(thislist)	['banana', 'cherry']

Deleting entire list

Code	Output
thislist = ["apple", "banana", "cherry"] del thislist	Nothing will be printed

cleaning the list

Code	Output
thislist = ["apple", "banana", "cherry"] thislist.clear() print(thislist)	[]

for loop in list

Code	Output
thislist = ["apple", "banana", "cherry"]	apple banana

`for x in thislist:` `  print(x)`	cherry

Using range and len method in list

Code	Output

`thislist =`
`["apple", "banana", "cherry"]`
`for i in range(len(thislist)):`
`  print(thislist[i])`
	apple
	banana
	cherry

while loop in list

Code	Output

`thislist =`
`["apple", "banana", "cherry"]`
`i = 0`
`while i < len(thislist):`
`  print(thislist[i])`
`  i = i + 1`
	apple
	banana
	cherry

Looping Using List Comprehension

Code	Output

`thislist =`
`["apple", "banana", "cherry"]`
`[print(x) for x in thislist]`
	apple
	banana
	cherry

Using for loop in List Comprehension

Code	Output

`fruits =`
`["apple", "banana", "cherry", "kiwi", "mango"]`
`newlist = []`
`for x in fruits:`
`  if "a" in x:`
`  newlist.append(x)`
`print(newlist)`
	`['apple', 'banana', 'mango']`

Without List Comprehension

Code	Output

`fruits =`
`["apple", "banana", "cherry", "kiwi", "mango"]`
`newlist = [x for x in fruits if "a" in x]`
`print(newlist)`
	`['apple', 'banana', 'mango']`

Only accept items that are not present in the element provided

Code	Output

`fruits = ["apple", "banana", "cherry", "kiwi", "mango"]`
`newlist = [x for x in fruits if x != "apple"]`
`print(newlist)`
	`['banana', 'cherry', 'kiwi', 'mango']`

List using no if statement

Code	Output

`fruits = ["apple", "banana", "cherry", "kiwi", "mango"]`
`newlist = [x for x in fruits]`
`print(newlist)`
	`['apple', 'banana', 'cherry', 'kiwi', 'mango']`

Iterable in list

Code	Output
`newlist = [x for x in range(10)]`	`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

Accept only numbers lower than mentioned number of a list

Code	Output
`newlist = [x for x in range(10) if x < 5]`	`[0, 1, 2, 3, 4]`

Set the values in the new list to upper case of a list

Code	Output
`fruits = ["apple", "banana", "cherry", "kiwi", "mango"]` `newlist = [x.upper() for x in fruits]`	`['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']`

Set all values in the new list to 'hello':

Code	Output
`fruits = ["apple", "banana", "cherry", "kiwi", "mango"]` `newlist = ['hello' for x in fruits]`	`['hello', 'hello', 'hello', 'hello', 'hello']`

Return "orange" instead of "banana" of a list

Code	Output
fruits = ["apple", "banana", "cherry", "kiwi", "mango"] newlist = [x if x != "banana" else "orange" for x in fruits]	['apple', 'orange', 'cherry', 'kiwi', 'mango']

Using sort in list

Code	Output
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"] thislist.sort() print(thislist)	['banana', 'kiwi', 'mango', 'orange', 'pineapple']

Sort the list numerically

Code	Output
thislist = [100, 50, 65, 82, 23] thislist.sort() print(thislist)	['banana', 'kiwi', 'mango', 'orange', 'pineapple']

Sort list in descending order

Code	Output
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"] thislist.sort(reverse = True) print(thislist)	['pineapple', 'orange', 'mango', 'kiwi', 'banana']

Sort the list descending numerically

Code	Output
thislist = [100, 50, 65, 82, 23] thislist.sort(reverse = True) print(thislist)	[100, 82, 65, 50, 23]

Customize sort function in list

Code	Output
def myfunc(n): return abs(n - 50) thislist = [100, 50, 65, 82, 23] thislist.sort(key = myfunc) print(thislist)	[50, 65, 23, 82, 100]

Case sensitive sorting in list

Code	Output

thislist = ["banana", "Orange", "Kiwi", "cherry"] thislist.sort() print(thislist)	['Kiwi', 'Orange', 'banana', 'cherry']

Case insensitive sorting in list

Code	Output
thislist = ["banana", "Orange", "Kiwi", "cherry"] thislist.sort(key = str.lower) print(thislist)	['banana', 'cherry', 'Kiwi', 'Orange']

reverse method in list

Code	Output
thislist = ["banana", "Orange", "Kiwi", "cherry"] thislist.reverse() print(thislist)	['cherry', 'Kiwi', 'Orange', 'banana']

Copy a list

Code	Output
thislist = ["apple", "banana", "cherry"] mylist = thislist.copy() print(mylist)	['apple', 'banana', 'cherry']

Using list method

Code	Output
thislist = ["apple", "banana", "cherry"] mylist = list(thislist) print(mylist)	['apple', 'banana', 'cherry']

Join two list

Code	Output
list1 = ["a", "b", "c"] list2 = [1, 2, 3] list3 = list1 + list2 print(list3)	['a', 'b', 'c', 1, 2, 3]

append method in list

Code	Output

list1 = ["a", "b" , "c"] list2 = [1, 2, 3] for x in list2: list1.append(x) print(list1)	['a', 'b', 'c', 1, 2, 3]

extend method in list

Code	Output
list1 = ["a", "b" , "c"] list2 = [1, 2, 3] list1.extend(list2) print(list1)	['a', 'b', 'c', 1, 2, 3]

clear method in list

Code	Output
fruits = ['apple', 'banana', 'cherry', 'orange'] fruits.clear()	[]

count method in list

Code	Output
fruits = ['apple', 'banana', 'cherry'] x = fruits.count("cherry")	1

Creating tuple

Code	Output
thistuple = ("apple", "banana", "cherry") print(thistuple)	('apple', 'banana', 'cherry')

Length of Tuple

Code	Output
thistuple = ("apple", "banana", "cherry") print(len(thistuple))	3

Create Tuple with One Item

Code	Output
thistuple = ("apple",) print(type(thistuple)) #NOT a tuple	<class 'tuple'> <class 'str'>

thistuple = ("apple") print(type(thistuple))	

Data Type of Tuple Item

Code	Output
tuple1 = ("apple", "banana", "cherry") tuple2 = (1, 5, 7, 9, 3) tuple3 = (True, False, False)	<class 'tuple'> <class 'tuple'> <class 'tuple'>

Mixed Data Type Tuple

Code	Output
tuple1 = ("abc", 34, True, 40, "male")	<class 'tuple'>

type method in tuple

Code	Output
mytuple = ("apple", "banana", "cherry") print(type(mytuple))	<class 'tuple'>

tuple constructor in tuple

Code	Output
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets print(thistuple)	('apple', 'banana', 'cherry')

Accessing Tuple

Code	Output
thistuple = ("apple", "banana", "cherry") print(thistuple[1])	banana

Negative Indexing in Tuple

Code	Output
thistuple = ("apple", "banana", "cherry") print(thistuple[-1])	cherry

Range of Index in tuple

Code	Output
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") print(thistuple[2:5])	('cherry', 'orange', 'kiwi')

Printing first range index in tuple

Code	Output
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") print(thistuple[:4])	('apple', 'banana', 'cherry', 'orange')

Printing after a range index in tuple

Code	Output
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") print(thistuple[2:])	('cherry', 'orange', 'kiwi', 'melon', 'mango')

Range of Negative Indexes in tuple

Code	Output
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") print(thistuple[-4:-1])	('orange', 'kiwi', 'melon')

Check if Item Exists\ Using in keyword in tuple

Code	Output
thistuple = ("apple", "banana", "cherry") if "apple" in thistuple: print("Yes, 'apple' is in the fruits tuple")	Yes, 'apple' is in the fruits tuple

Change tuple value

Code	Output
x = ("apple", "banana", "cherry") y = list(x) y[1] = "kiwi"	('apple', 'kiwi', 'cherry')

x = tuple(y) print(x)	

Convert tuple to list

Code	Output
thistuple = ("apple", "banana", "cherry") y = list(thistuple) y.append("orange") thistuple = tuple(y)	('apple', 'banana', 'cherry', 'orange')

Add tuple to tuple

Code	Output
thistuple = ("apple", "banana", "cherry") y = ("orange",) thistuple += y print(thistuple)	('apple', 'banana', 'cherry', 'orange')

Remove Item

Code	Output
thistuple = ("apple", "banana", "cherry") y = list(thistuple) y.remove("apple") thistuple = tuple(y)	('banana', 'cherry')

Using delete in tuple

Code	Output
thistuple = ("apple", "banana", "cherry") del thistuple print(thistuple) #this will raise an error because the tuple no longer exists	NameError: name 'thistuple' is not defined

Unpacking a tuple

Code	Output
fruits = ("apple", "banana", "cherry") (green, yellow, red) = fruits print(green)	apple banana cherry

`print(yellow)` `print(red)`	

Using Asterisk*

Code	Output

`fruits =`
`("apple", "banana", "cherry", "strawberry", "raspberry")`
`(green, yellow, *red) = fruits`
`print(green)`
`print(yellow)`
`print(red)`
	`apple`
	`banana`
	`['cherry',`
	`'strawberry',`
	`'raspberry']`

Add list of values in tuple

Code	Output

`fruits =`
`("apple", "mango", "papaya", "pineapple", "cherry")`
`(green, *tropic, red) = fruits`
`print(green)`
`print(tropic)`
`print(red)`
	`apple`
	`['mango', 'papaya',`
	`'pineapple']`
	`cherry`

for loop in tuple

Code	Output

`thistuple =`
`("apple", "banana", "cherry")`
`for x in thistuple:`
`  print(x)`
	`apple`
	`banana`
	`cherry`

using range and length in tuple

Code	Output

`thistuple =`
`("apple", "banana", "cherry")`
`for i in range(len(thistuple)):`
`  print(thistuple[i])`
	`apple`
	`banana`
	`cherry`

while loop in tuple

Code	Output

`thistuple =`
`("apple", "banana", "cherry")`
`i = 0`
`while i < len(thistuple):`
`  print(thistuple[i])`
`  i = i + 1`
	`apple`
	`banana`
	`cherry`

Join two tuples

Code	Output

`tuple1 = ("a", "b" , "c")`
`tuple2 = (1, 2, 3)`
`tuple3 = tuple1 + tuple2`
`print(tuple3)`
	`('a', 'b', 'c', 1, 2, 3)`

Multiple two tuples

Code	Output

`fruits =`
`("apple", "banana", "cherry")`
`mytuple = fruits * 2`
`print(mytuple)`
	`('apple', 'banana', 'cherry',`
	`'apple', 'banana', 'cherry')`

Using count method in tuple

Code	Output

`thistuple =`
`(1, 3, 7, 8, 7, 5, 4, 6, 8, 5)`
`x = thistuple.count(5)`
`print(x)`
	`2`

Creating Set

Code	Output

`thisset =`
`{"apple", "banana", "cherry"}`
`print(len(thisset))`
	`3`

Data Type of set item

Code	Output

`set1 =`
`{"apple", "banana", "cherry"}`
`set2 = {1, 5, 7, 9, 3}`
`set3 = {True, False, False}`
	`<class 'set'>`
	`<class 'set'>`
	`<class 'set'>`

Mixed Data Type Set

Code	Output
`set1 =` `{"abc", 34, True, 40, "male"}`	`<class 'set'>`

type method in set

Code	Output

myset = {"apple", "banana", "cherry"} print(type(myset))	<class 'set'>

set method in set

Code	Output
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets print(thisset)	{'apple', 'cherry', 'banana'}

Accessing Set

Code	Output
thisset = {"apple", "banana", "cherry"} for x in thisset: print(x)	cherry banana apple

Add Items to set

Code	Output
thisset = {"apple", "banana", "cherry"} thisset.add("orange") print(thisset)	{'orange', 'banana', 'apple', 'cherry'}

Add sets

Code	Output
thisset = {"apple", "banana", "cherry"} tropical = {"pineapple", "mango", "papaya"} thisset.update(tropical) print(thisset)	{'apple', 'cherry', 'mango', 'papaya', 'banana', 'pineapple'}

update method in set

Code	Output
thisset = {"apple", "banana", "cherry"} mylist = ["kiwi", "orange"] thisset.update(mylist) print(thisset)	{'apple', 'orange', 'cherry', 'banana', 'kiwi'}

Remove items from a set

Code	Output
thisset = {"apple", "banana", "cherry"} thisset.remove("banana") print(thisset)	{'apple', 'cherry'}

discard method in set

Code	Output
thisset = {"apple", "banana", "cherry"} thisset.discard("banana") print(thisset)	{'cherry', 'apple'}

pop method in set

Code	Output
thisset = {"apple", "banana", "cherry"} x = thisset.pop() print(x) print(thisset)	banana {'cherry', 'apple'}

clear method in set

Code	Output
thisset = {"apple", "banana", "cherry"} thisset.clear() print(thisset)	set()

delete method in set

Code	Output
thisset = {"apple", "banana", "cherry"} del thisset print(thisset)	NameError: name 'thisset' is not defined

Using for loop in set

Code	Output
thisset = {"apple", "banana", "cherry"} for x in thisset: print(x)	apple cherry banana

union method in set

Code	Output

`set1 = {"a", "b" , "c"}`
`set2 = {1, 2, 3}`
`set3 = set1.union(set2)`
`print(set3)`
	`{'a', 1, 2, 3, 'b', 'c'}`

update method in set

Code	Output

`set1 = {"a", "b" , "c"}`
`set2 = {1, 2, 3}`
`set1.update(set2)`
`print(set1)`
	`{1, 'c', 2, 3, 'a', 'b'}`

intersection_update method in set

Code	Output

`x = {"apple", "banana", "cherry"}`
`y =`
`{"google", "microsoft", "apple"}`
`x.intersection_update(y)`
`print(x)`
	`{'apple'}`

intersection method in set

Code	Output

`x = {"apple", "banana", "cherry"}`
`y =`
`{"google", "microsoft", "apple"}`
`z = x.intersection(y)`
`print(z)`
	`{'apple'}`

symmetric_difference_update in set

Code	Output

`x = {"apple", "banana", "cherry"}`
`y =`
`{"google", "microsoft", "apple"}`
`x.symmetric_difference_update(y)`
`print(x)`
	`{'google', 'banana', 'cherry',`
	`'microsoft'}`

symmetric_difference in set

Code	Output
`x = {"apple", "banana", "cherry"}` `y =`	`{'microsoft', 'banana', 'cherry',` `'google'}`

{"google", "microsoft", "apple"} z = x.symmetric_difference(y) print(z)	

Copy method in set

Code	Output
fruits = {"apple", "banana", "cherry"} x = fruits.copy() print(x)	{'banana', 'apple', 'cherry'}

Set difference() Method in set

Code	Output
x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.difference(y) print(z)	{'cherry', 'banana'}

difference_update() method in set

Code	Output
x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} x.difference_update(y) print(x)	{'banana', 'cherry'}

isdisjoint() Method in set

Code	Output
x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "facebook"} z = x.isdisjoint(y) print(z)	True

issuperset() Method in set

Code	Output
x = {"f", "e", "d", "c", "b", "a"} y = {"a", "b", "c"} z = x.issuperset(y) print(z)	True

Create and print dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
	{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

print value of an attribute

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
	Ford

Removing duplicates from dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
	{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

Dictionary length

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(len(thisdict))
	3

Datatype of dictionary item

Code	Output

thisdict = {
  "brand": "Ford",
  "electric": False,
	<class 'dict'>

"year": 1964, "colors": ["red", "white", "blue"] } print(type(thisdict))	

Accessing items of dictionary

Code	Output

thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = thisdict["model"]
	Mustang

Using Get method in dictionary

Code	Output

thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = thisdict.get("model")
	Mustang

Using Keys method in dictionary

Code	Output

thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = thisdict.keys()
	dict_keys(['brand', 'model',
	'year'])

Add a new item and the keys in a dictionary

Code	Output

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.keys()
print(x) #before the change
car["color"] = "white"
print(x) #after the change
	dict_keys(['brand', 'model',
	'year'])
	dict_keys(['brand', 'model',
	'year', 'color'])

Using values method in dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.values()
	dict_values(['Ford', 'Mustang', 1964])

Update values in dictionary

Code	Output

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.values()
print(x) #before the change
car["year"] = 2020
print(x) #after the change
	dict_values(['Ford', 'Mustang', 1964])
	dict_values(['Ford', 'Mustang', 2020])

Add a new item in dictionary

Code	Output

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.values()
print(x) #before the change
car["color"] = "red"
print(x) #after the change
	dict_values(['Ford', 'Mustang', 1964])
	dict_values(['Ford', 'Mustang', 1964, 'red'])

Get a list of the key:value pairs in dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.items()
	dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

Make a change in the original dictionary

Code	Output

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.items()
print(x) #before the change
car["year"] = 2020
print(x) #after the change
	dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
	dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2020)])

Add a new item to the original dictionary

Code	Output

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
 	
x = car.items()
print(x) #before the change
car["color"] = "red"
print(x) #after the change
	dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
	dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red')])

Using “if” “in” in dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
	Yes, 'model' is one of the keys in the thisdict dictionary

Change Values in Dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
	{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

Changing Value in Dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
	{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

Adding Item in Dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
	{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

Updating Item in Dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})
	{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

pop method in dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
	{'brand': 'Ford', 'year': 1964}

Pop item method in dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
	{'brand': 'Ford', 'model': 'Mustang'}

`  "year": 1964`
`}`
`thisdict.popitem()`
`print(thisdict)`	

delete method in dictionary

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`del thisdict["model"]`
`print(thisdict)`
	`{'brand': 'Ford', 'year': 1964}`

Error in dictionary

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`del thisdict`
`print(thisdict) #this will cause an error because "thisdict" no longer exists.`
	`thisdict = {`
	`  "brand": "Ford",`
	`  "model": "Mustang",`
	`  "year": 1964`
	`}`
	`print (thisdict)`
	`del thisdict`

Output: `NameError: name 'thisdict' is not defined`	**Output:** `{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}`

clear method in dictionary

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`thisdict.clear()`
`print(thisdict)`
	`{}`

Print all keys of dictionary using for loop

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
	`brand`
	`model`
	`year`

`  "year": 1964`
`}`
`for x in thisdict:`
`  print(x)`	

Print all values of dictionary using for loop

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`for x in thisdict:`
`  print(thisdict[x])`
	`Ford`
	`Mustang`
	`1964`

Using values method in dictionary using for loop

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`for x in thisdict.values():`
`  print(x)`
	`Ford`
	`Mustang`
	`1964`

Using keys method in dictionary using for loop

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`for x in thisdict.keys():`
`  print(x)`
	`brand`
	`model`
	`year`

Using keys and values method in dictionary using for loop

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`for x, y in thisdict.items():`
`  print(x, y)`
	`brand Ford`
	`model Mustang`
	`year 1964`

Copy a dictionary

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
	{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

Copy a dictionary in a function

Code	Output

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
	{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

Create a dictionary containing three dictionaries

Code	Output

myfamily = {
  "child1" : {
  "name" : "Emil",
  "year" : 2004
  },
  "child2" : {
  "name" : "Tobias",
  "year" : 2007
  },
  "child3" : {
  "name" : "Linus",
  "year" : 2011
  }
}
	{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}

Create three dictionary containing one dictionary

Code	Output

child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
	{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}

```
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

Using fromkeys in dictionary

Code	Output

`thisdict = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`x = ('key1', 'key2', 'key3')`
`y = 0`
`thisdict = dict.fromkeys(x, y)`
`print(thisdict)`
	`{'key1': 0, 'key2': 0, 'key3': 0}`

Using setdefault in dictionary

Code	Output

`car = {`
`  "brand": "Ford",`
`  "model": "Mustang",`
`  "year": 1964`
`}`
`x =`
`car.setdefault("model", "Bronco")`
`print(x)`
	`Mustang`

Create a set in dictionary

Code	Output

`thisset =`
`{"apple", "banana", "cherry"}`
`print(thisset)`
`#Length of a set`
	`{'banana', 'cherry', 'apple'}`
	`3`

thisset = {"apple", "banana", "cherry"} print(len(thisset))	

If

Code	Output
a = 33 b = 200 if b > a: print("b is greater than a")	b is greater than a

Error in if

a = 33 b = 200 if b > a: print("b is greater than a")	a = 33 b = 200 if b > a: print("b is greater than a")
Output: IndentationError: expected an indented block	**Output:** b is greater than a

Elif

Code	Output

a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
	a and b are equal

Else

Code	Output

a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
	a is greater than b

If...Else

Code	Output
a = 200 b = 33	b is not greater than a

if b > a:
 print("b is greater than a")
else:
 print("b is not greater than a")	

One Line if Statement

Code	Output
a = 200 b = 33 if a > b: print("a is greater than b")	a is greater than b

One Line if...else Statement

Code	Output
a = 2 b = 330 print("A") if a > b else print("B")	B

One Line if Statement with 3 conditions:

Code	Output
a = 330 b = 330 print("A") if a > b else print("=") if a == b else print("B")	=

if using logical AND

Code	Output
a = 200 b = 33 c = 500 if a > b and c > a: print("Both conditions are True")	Both conditions are True

if using logical OR

Code	Output
a = 200 b = 33 c = 500 if a > b or a > c: print("At least one of the conditions is True")	At least one of the conditions is True

Nested If

Code	Output

x = 41
if x > 10:
  print("Above ten,")
  if x > 20:
  print("and also above 20!")
  else:
  print("but not above 20.")
	Above ten,
	and also above 20!

pass

Code	Output
a = 33 b = 200 if b > a: pass	

while

Code	Output

i = 1
while i < 6:
  print(i)
  i += 1
	1
	2
	3
	4
	5

break statement

Code	Output

i = 1
while i < 6:
  print(i)
  if i == 3:
  break
  i += 1
	1
	2
	3

continue statement

Code	Output

i = 0
while i < 6:
  i += 1
  if i == 3:
  continue
  print(i)
	1
	2
	4
	5
	6

else statement

Code	Output

i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
	1
	2
	3
	4
	5
	i is no longer less than 6

for

Code	Output
fruits = ["apple", "banana", "cherry"] for x in fruits: print(x)	apple banana cherry

for with string

Code	Output
fruits = ["apple", "banana", "cherry"]for x in "banana": print(x)	b a n a n a

break with string

fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
  break
	fruits = ["apple", "banana", "cherry"]
	for x in fruits:
	  if x == "banana":
	  break
	  print(x)

Output: apple banana	**Output:** apple

Creating Function

Code	Output
def my_function(): print("Hello from a function") print (my_function())	Hello from a function None

Calling a function

Code	Output

`def my_function():`
`  print("Hello from a function")`
 	
`my_function()`
	Hello from a function

Passing argument to function

Code	Output

`def my_function(fname):`
`  print(fname + " Refsnes")`
 	
`my_function("Emil")`
`my_function("Tobias")`
`my_function("Linus")`
	Emil Refsnes
	Tobias Refsnes
	Linus Refsnes

Number of arguments

Code	Output

`def my_function(fname, lname):`
`  print(fname + " " + lname)`
 	
`my_function("Emil", "Refsnes")`
	Emil Refsnes

Error in function

`def my_function(fname, lname):`
`  print(fname + " " + lname)`
 	
`my_function("Emil")`
	def my_function(fname, lname):
	  print(fname + " " + lname)
	my_function("fname", "lname")

Output: TypeError: my_function() missing 1 required positional argument: 'lname'	**Output:** fname lname

Arbitrary Arguments

Code	Output

`def my_function(*kids):`
`  print("The youngest child is " + kids[2])`
 	
`my_function("Emil", "Tobias", "Linus")`
	The youngest child is Linus

Keyword Arguments

Code	Output

`def my_function(child3, child2, child1):`
`  print("The youngest child is " + child3)`
 	
`my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")`
	The youngest child is Linus

Arbitrary Keyword Arguments

Code	Output

`def my_function(**kid):`
`  print("His last name is " + kid["lname"])`
 	
`my_function(fname = "Tobias", lname = "Refsnes")`
	His last name is Refsnes

Default Parameter Value

Code	Output

`def my_function(country = "Norway"):`
`  print("I am from " + country)`
 	
`my_function("Sweden")`
`my_function("India")`
`my_function()`
`my_function("Brazil")`
	I am from Sweden
	I am from India
	I am from Norway
	I am from Brazil

Passing a List as an Argument

Code	Output

`def my_function(food):`
`  for x in food:`
`  print(x)`
`fruits = ["apple", "banana", "cherry"]`
 	
`my_function(fruits)`
	apple
	banana
	cherry

Return Values

Code	Output

`def my_function(x):`
`  return 5 * x`
 	
`print(my_function(3))`
	15
	25
	45

`print(my_function(5))` `print(my_function(9))`	

pass statement in function

Code	Output

`def myfunction():`
`  pass`
`print(my_function())`
	None

Recursion

Code	Output

`def tri_recursion(k):`
`  if(k > 0):`
`  result = k + tri_recursion(k - 1)`
`  print(result)`
`  else:`
`  result = 0`
`  return result`
 	
`print("\n\nRecursion Example Results")`
`tri_recursion(6)`
	Recursion Example Results
	1
	3
	6
	10
	15
	21

Creating Array

Code	Output
`cars = ["Ford", "Volvo", "BMW"]` `print (cars)`	['Ford', 'Volvo', 'BMW']

Accessing first array item

Code	Output

`cars = ["Ford", "Volvo", "BMW"]`
`x = cars[0]`
`print (cars)`
	['Ford', 'Volvo', 'BMW']

Modifying first array item

Code	Output

`cars = ["Ford", "Volvo", "BMW"]`
`cars[0] = "Toyota"`
`print (cars)`
	['Toyota', 'Volvo', 'BMW']

Length of an Array

Code	Output

`cars = ["Ford", "Volvo", "BMW"]`
`x = len(cars)`
`#for in array`
`for x in cars:`
`  print(x)`
	Ford
	Volvo
	BMW

Adding Array Elements

Code	Output

`cars = ["Ford", "Volvo", "BMW"]`
`cars.append("Honda")`
`print (cars)`
	['Ford', 'Volvo', 'BMW', 'Honda']

Removing Array Elements

Code	Output

`cars = ["Ford", "Volvo", "BMW"]`
`cars.remove("Volvo")`
`print (cars)`
	['Ford', 'BMW']

Pop in Array Elements

Code	Output

`cars = ["Ford", "Volvo", "BMW"]`
`cars.pop(1)`
`print (cars)`
	['Ford', 'BMW']

Clear Array Elements

Code	Output

`fruits =`
`['apple', 'banana', 'cherry', 'orange']`
`fruits.clear()`
	[]

Copying Array Elements

Code	Output

`fruits =`
`['apple', 'banana', 'cherry', 'orange']`
`x = fruits.copy()`
	['apple', 'banana', 'cherry', 'orange']

Counting Array Elements

Code	Output

fruits = ['apple', 'banana', 'cherry'] x = fruits.count("cherry")	1

Extending Array Elements

Code	Output
fruits = ['apple', 'banana', 'cherry'] cars = ['Ford', 'BMW', 'Volvo'] fruits.extend(cars)	['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']

Indexing Array Elements

Code	Output
fruits = ['apple', 'banana', 'cherry'] x = fruits.index("cherry")	2

Inserting Array Elements

Code	Output
fruits = ['apple', 'banana', 'cherry'] fruits.insert(1, "orange")	['apple', 'orange', 'banana', 'cherry']

Reversing Array Elements

Code	Output
fruits = ['apple', 'banana', 'cherry'] fruits.reverse()	['cherry', 'banana', 'apple']

Sorting Array Elements

Code	Output
cars = ['Ford', 'BMW', 'Volvo'] cars.sort()	['BMW', 'Ford', 'Volvo']

Iterator in tuple

Code	Output
mytuple = ("apple", "banana", "cherry") myit = iter(mytuple) print(next(myit)) print(next(myit)) print(next(myit))	apple banana cherry

Iterator in String

Code	Output

`mystr = "banana"`
`myit = iter(mystr)`
`print(next(myit))`
`print(next(myit))`
`print(next(myit))`
`print(next(myit))`
`print(next(myit))`
`print(next(myit))`
	b
	a
	n
	a
	n
	a

Iterator in tuple using for loop

Code	Output

`mytuple = ("apple", "banana", "cherry")`
`for x in mytuple:`
`  print(x)`
	apple
	banana
	cherry

Iterator in string using for loop

Code	Output

`mystr = "banana"`
`for x in mystr:`
`  print(x)`
	b
	a
	n
	a
	n
	a

Local Scope

Code	Output

`def myfunc():`
`  x = 300`
`  print(x)`
`myfunc()`
	300

Local Variable from a function

Code	Output

`def myfunc():`
`  x = 300`
`  def myinnerfunc():`
`  print(x)`
`  myinnerfunc()`
`myfunc()`
	300

Global Scope

Code	Output
x = 300 def myfunc(): print(x) myfunc() print(x)	300 300

Naming Variable

Code	Output

x = 300
def myfunc():
  x = 200
  print(x)
myfunc()
print(x)
	200
	300

Global Keyword

Code	Output

def myfunc():
  global x
  x = 300
myfunc()
print(x)
	300

Global Variable from a function

Code	Output
x = 300 def myfunc(): global x x = 200 myfunc() print(x)	200

Creating Module

Code	Output

def greeting(name):
  print("Hello, " + name)
Variables in Module
person1 = {
  "name": "John",
  "age": 36,	

`  "country": "Norway"` `}`	

Import Module

Code	Output

`import mymodule`
`a = mymodule.person1["age"]`
`print(a)`	

Renaming Module

Code	Output

`import mymodule as mx`
`a = mx.person1["age"]`
`print(a)`	

Importing Built-in Module

Code	Output

`import platform`
`x = platform.system()`
`print(x)`	

Importing Module using function

Code	Output

`import platform`
`x = dir(platform)`
`print(x)`	

Importing Module from other Module

Code	Output

`def greeting(name):`
`  print("Hello, " + name)`
 	
`person1 = {`
`  "name": "John",`
`  "age": 36,`
`  "country": "Norway"`
`}`	

Import a module from dictionary

Code	Output
`from mymodule import person1` `print (person1["age"])`	

Importing camelcase module

Code	Output

`import camelcase`
`c = camelcase.CamelCase()`
`txt = "hello world"`
`print(c.hump(txt))`	

try

Code	Output

`try:`
`  print(x)`
`except:`
`  print("An exception occurred")`
	An exception occurred

Error in try exception handling

print(x)	x=10 print(x)
Output: NameError: name 'x' is not defined	**Output:** 10

Many Exception

Code	Output

`try:`
`  print(x)`
`except NameError:`
`  print("Variable x is not defined")`
`except:`
`  print("Something else went wrong")`
	Variable x is not defined

Using else in exception

Code	Output

`try:`
`  print("Hello")`
`except:`
`  print("Something went wrong")`
`else:`
`  print("Nothing went wrong")`
	Hello
	Nothing went wrong

Using finally in exception

Code	Output

`try:`
`  print(x)`
`except:`
`  print("Something went wrong")`
`finally:`
`  print("The 'try except' is finished")`
	Something went wrong
	The 'try except' is finished

Using try to open and write a file

Code	Output

`try:`
`  f = open("demofile.txt")`
`  try:`
`  f.write("Lorum Ipsum")`
`  except:`
`  print("Something went wrong when writing to the file")`
`  finally:`
`  f.close()`
`except:`
`  print("Something went wrong when opening the file")`
	Something went wrong when opening the file

Raise an exception

Code	Output

`x = -1`
`if x < 0:`
`  raise Exception("Sorry, no numbers below zero")`
	Exception: Sorry, no numbers below zero

Raise a TypeError exception

Code	Output

`x = "hello"`
`if not type(x) is int:`
`  raise TypeError("Only integers are allowed")`
	TypeError: Only integers are allowed

Different Exception Handling

Exception	Description	Code	Output

ArithmeticError	Raised when an error occurs in numeric calculations	`try:`
		`  a = int(input("Enter a:"))`
		`  b = int(input("Enter b:"))`
		`  if b is 0:`
		`  raise ArithmeticError`
			Enter a:10
			Enter b:0
			The value of b can't be 0

		else: print("a/b = ",a/b) except ArithmeticError: print("The value of b can't b e 0")	
AssertionError	Raised when an assert statement fails	# Handling it manually try: x = 1 y = 0 assert y != 0, "Invalid Operation" print(x / y) # the errror_message provided by the user gets printed except AssertionError as msg: print(msg)	Invalid Operation

AttributeError	Raised when attribute reference or assignment fails	# Python program to demonstrate
		# AttributeError
		X = 10
		# Raises an AttributeError
		X.append(6)
			AttributeError: 'int' object has no attribute 'append'

Exception	Base class for all exceptions	# Python program to handle simple runtime error #Python 3 a = [1, 2, 3] try: print ("Second element = %d" %(a[1])) # Throws error since there are only 3 elements in array print ("Fourth element = %d" %(a[3])) except: print ("An error occurred")	Second element = 2 An error occurred
EOFError	Raised when the input() method hits an "end of file" condition (EOF)	try: n = int(input()) print(n * 10) except EOFError as e: print(e)	5 50

FloatingPointError	Raised when a floating point calculation fails		
GeneratorExit	Raised when a generator is closed (with the close() method)		
ImportError	Raised when an imported module does not exist		
IndentationError	Raised when indendation is not correct	import sys try: s = {'a':5, 'b':7}['c'] except: print (sys.exc_info())	IndentationError: expected an indented block after 'try' statement on line 2
IndexError	Raised when an index of a sequence does not exist	num_list=[1,2,3,4] value=num_list[4] print (value)	IndexError: list index out of range
KeyError	Raised when a key does not exist in a dictionary		
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete		
LookupError	Raised when errors raised cant be found		
MemoryError	Raised when a program runs out of memory		
NameError	Raised when a variable does not exist	avg=total/10 #where total is not defined print (avg)	NameError: name 'total' is not defined
NotImplementedError	Raised when an abstract method requires an inherited class to override the method		
OSError	Raised when a system related operation causes an error		

OverflowError	Raised when the result of a numeric calculation is too large		
ReferenceError	Raised when a weak reference object does not exist		
RuntimeError	Raised when an error occurs that do not belong to any specific expections		
StopIteration	Raised when the next() method of an iterator has no further values		
SyntaxError	Raised when a syntax error occurs		
TabError	Raised when indentation consists of tabs or spaces		
SystemError	Raised when a system error occurs		
SystemExit	Raised when the sys.exit() function is called		
TypeError	Raised when two different types are combined		
UnboundLocalError	Raised when a local variable is referenced before assignment		
UnicodeError	Raised when a unicode problem occurs		
UnicodeEncodeError	Raised when a unicode encoding problem occurs		
UnicodeDecodeError	Raised when a unicode decoding problem occurs		
UnicodeTranslateError	Raised when a unicode translation problem occurs		

ValueError	Raised when there is a wrong value in a specified data type	try:
		  age = int(input("Enter the age:"))
		  if(age<18):
		  raise ValueError
		  else:
		  print("the age is valid")
		except ValueError:
		  print("The age is not valid")
			Enter the age:17
			The age is not valid

ZeroDivisionError	Raised when the second operator in a division is zero	num_list=[] total=0 avg=total/len(num_list)	ZeroDivisionError: division by zero

Using input

Code	Output
`username = input("Enter username:")` `print("Username is: " + username)`	`Enter username: Soumi` `Username is:  Soumi`

Using format in string

Code	Output

`price = 49`
`txt = "The price is {} dollars"`
`print(txt.format(price))`
	`The price is 49 dollars`

Format as a number with two decimals

Code	Output

`price = 49`
`txt = "The price is {:.2f} dollars"`
`print(txt.format(price))`
	`The price is 49.00 dollars`

formatting Multiple Values

Code	Output

`quantity = 3`
`itemno = 567`
`price = 49`
`myorder = "I want {} pieces of item number {} for {:.2f} dollars."`
`print(myorder.format(quantity, itemno, price))`
	`I want 3 pieces of item number 567 for 49.00 dollars.`

Using index number

Code	Output

quantity = 3 itemno = 567 price = 49 myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars." print(myorder.format(quantity, itemno, price))	I want 3 pieces of item number 567 for 49.00 dollars.

Using index number for same value

Code	Output
age = 36 name = "John" txt = "His name is {1}. {1} is {0} years old." print(txt.format(age, name))	His name is John. John is 36 years old.

Named Indexes

Code	Output
myorder = "I have a {carname}, it is a {model}." print(myorder.format(carname = "Ford", model = "Mustang"))	I have a Ford, it is a Mustang.

Difference between Set and Array

Set	Array
It is un-ordered	It is ordered
It does not allow duplicate elements	It allows duplicate elements
It is represented by []	It is represented by ()

Difference between Tuple, List and Dictionary

It is immutable (values cannot be changed)	It is mutable (values can be changed)	It is mutable (values can be changed)
It is represented by ()	It is represented by []	It is represented by {}
It is ordered	It is ordered	It is ordered
Methods used are count and index	Methods used are append, clear, copy, count, extend, index, insert, pop, remove, reverse, sort	Methods used are clear, copy, fromkeys, getitems, pop, popitems, setdefault, update, values etc

Difference between Yield and Return

YIELD	RETURN
Yield is generally used to convert a regular Python function into a generator.	Return is generally used for the end of the execution and “returns” the result to the caller statement.
It replace the return of a function to suspend its execution without destroying local variables.	It exits from a function and handing back a value to its caller.
It is used when the generator returns an intermediate result to the caller.	It is used when a function is ready to send a value.
Code written after yield statement execute in next function call.	while, code written after return statement wont execute.
It can run multiple times.	It only runs single time.
Yield statement function is executed from the last state from where the function get paused.	Every function calls run the function from the start.

Difference between Break and Continue

BASIS FOR COMPARISON	BREAK	CONTINUE
Task	It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
Control after break/continue	'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
Causes	It causes early termination of loop.	It causes early execution of the next iteration.
Continuation	'break' stops the continuation of loop.	'continue' do not stops the continuation of loop, it only stops the current iteration.
Other uses	'break' can be used with 'switch', 'label'.	'continue' can not be executed with 'switch' and 'labels'.

Difference between Recursion and Iteration

BASIS FOR COMPARISON	RECURSION	ITERATION
Basic	The statement in a body of function calls the function itself.	Allows the set of instructions to be repeatedly executed.
Format	In recursive function, only termination condition (base case) is specified.	Iteration includes initialization, condition, execution of statement within loop and update (increments and decrements) the control variable.
Termination	A conditional statement is included in the body of the function to force the	The iteration statement is repeatedly executed until a certain condition is reached.

	function to return without recursion call being executed.	
Condition	If the function does not converge to some condition called (base case), it leads to infinite recursion.	If the control condition in the iteration statement never become false, it leads to infinite iteration.
Infinite Repetition	Infinite recursion can crash the system.	Infinite loop uses CPU cycles repeatedly.
Applied	Recursion is always applied to functions.	Iteration is applied to iteration statements or "loops".
Stack	The stack is used to store the set of new local variables and parameters each time the function is called.	Does not uses stack.
Overhead	Recursion possesses the overhead of repeated function calls.	No overhead of repeated function call.
Speed	Slow in execution.	Fast in execution.
Size of Code	Recursion reduces the size of the code.	Iteration makes the code longer.

Difference between If-Else and Switch

BASIS FOR COMPARISON	IF-ELSE	SWITCH
Basic	Which statement will be executed depend upon the output of the expression inside if statement.	Which statement will be executed is decided by user.
Expression	if-else statement uses multiple statement for multiple choices.	switch statement uses single expression for multiple choices.
Testing	if-else statement test for equality as well as for logical expression.	switch statement test only for equality.
Evaluation	if statement evaluates integer, character, pointer or floating-point type or boolean type.	switch statement evaluates only character or integer value.
Sequence of Execution	Either if statement will be executed or else statement is executed.	switch statement execute one case after another till a break statement is appeared or the end of switch statement is reached.
Default Execution	If the condition inside if statements is false, then by default the else statement is executed if created.	If the condition inside switch statements does not match with any of cases, for that instance the default statements is executed if created.
Editing	It is difficult to edit the if-else statement, if the nested if-else statement is used.	It is easy to edit switch cases as, they are recognized easily

Difference between While and Do-While

BASIS FOR COMPARISON	WHILE	DO-WHILE

General Form	while (condition) {
	statements; //body of loop
	}
		do{
		.
		statements; // body of loop.
		.
		} while(Condition);

Controlling Condition	In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
Iterations	The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.
Alternate name	Entry-controlled loop	Exit-controlled loop
Semi-colon	Not used	Used at the end of the loop

Difference between For and While

BASIS FOR COMPARISON	FOR	WHILE

Declaration	for(initialization; condition; iteration){
	//body of 'for' loop
	}
		while (condition) {
		statements; //body of loop
		}

Format	Initialization, condition checking, iteration statement are written at the top of the loop.	Only initialization and condition checking is done at the top of the loop.
Use	The 'for' loop used only when we already knew the number of iterations.	The 'while' loop used only when the number of iteration are not exactly known.
Condition	If the condition is not put up in 'for' loop, then loop iterates infinite times.	If the condition is not put up in 'while' loop, it provides compilation error.
Initialization	In 'for' loop the initialization once done is never repeated.	In while loop if initialization is done during condition checking, then initialization is done each time the loop iterate.
Iteration statement	In 'for' loop iteration statement is written at top, hence, executes only after all statements in loop are executed.	In 'while' loop, the iteration statement can be written anywhere in the loop.

Difference between pass and continue

Pass	continue
does nothing	jumps for next iteration
required when syntactically needed but practically not	required when want to skip the execution of remaining statement(s) inside the loop for current iteration
can be used as a placeholder for future code	can not be used as a placeholder for future code

Difference between global and local variable

Global Variable	Local Variable
Global variables are declared outside all the function blocks.	Local Variables are declared within a function block.
The scope remains throughout the program.	The scope is limited and remains within the function only in which they are declared.
Any change in global variable affects the whole program, wherever it is being used.	Any change in the local variable does not affect other functions of the program.
A global variable exists in the program for the entire time the program is executed.	A local variable is created when the function is executed, and once the execution is finished, the variable is destroyed.
It can be accessed throughout the program by all the functions present in the program.	It can only be accessed by the function statements in which it is declared and not by the other functions.
If the global variable is not initialized, it takes zero by default.	If the local variable is not initialized, it takes the garbage value by default.
Global variables are stored in the data segment of memory.	Local variables are stored in a stack in memory.
We cannot declare many variables with the same name.	We can declare various variables with the same name but in other functions.

Differences between Compilation and Interpretation in terms of advantages and disadvantages

Point	COMPILATION	INTERPRETATION
ADVANTAGES	i. the execution of the translated code is usually faster; ii. only the user has to have the compiler - the end-user may use the code without it; iii. the translated code is stored using machine language - as it is very hard to understand it, your own inventions and programming tricks are likely to remain your secret.	i. you can run the code as soon as you complete it - there are no additional phases of translation; ii. the code is stored using programming language, not machine language - this means that it can be run on computers using different machine languages; you don't compile your code separately for each different architecture.

Point	COMPILATION	INTERPRETATION
DISADVANTAGES	i. the compilation itself may be a very time-consuming process - you may not be able to run your code immediately after making an amendment; ii. you have to have as many compilers as hardware platforms you want your code to be run on.	i. don't expect interpretation to ramp up your code to high speed - your code will share the computer's power with the interpreter, so it can't be really fast; ii. both you and the end user have to have the interpreter to run your code.

Point to be Noted

Points	Definition\Explanation
Python	It’s a free, open source, interpreted, high level, object oriented language
Python is developed by	Guido Van Rossum
The name Python came from	Monty Python’s Flying Circus
IDLE	Integrated Development and Learning Environment
REPL	Read, Evaluate, Print, Loop
Instruction List	A complete set of known commands.
Machine Language	Computer’s language
Source Code	A program written in a high level programming language
Translator\Compilation	A computer program which directly executes instruction written in a programming language
Interpretation	You (or any user of the code) can translate the source program each time it has to be run; the program performing this kind of transformation is called an interpreter, as it interprets the code every time it is intended to be executed; it also means that you cannot just distribute the source code as-is, because the end-user also needs the interpreter to execute it.
CPython	A superset of the Python programming language. It is the default, reference implementation of Python written in C language.
Debugger	A tool that lets you launch your code step-by-step and inspect it at each moment of execution

Positional Parameter	It determines the position
Escape Character	Owes its name to the fact that is changes the next to it
Keyword	A word that cannot be used as a function name
Function	It starts with the keyword def. It must be placed before the first invocation.
Function Parameter	It is a kind of variable accessible only inside the function
Ordered	A way of passing arguments in which the order of the arguments determines the initial parameters values
Script	It is a text file that contains instructions which make up a Python program.
Scripting Languages	Languages designed to be utilized in the interpretation manner.
Language	Language is the keyword.
Machine Code	It is a low level programming language consisting of binary digits/bits that the computer read and understands
Source File	A file containing a program written in a high level programming language
Editor	It supports you in writing the code (it should have some special features, not available in simple tools); this dedicated editor will give you more than the standard OS equipment
Console	A command line interpreter which lets you to interact with OS and execute Python command and scripts
Keyword Parameter	It is determined by the arguments name specified along with its value
Reversed Word	Keywords is defined by Python’s lexis and is known as a reverse word
UNICODE	Is a standard for encoding and handling texts
ASCII	The ascii() function returns a readable version of any object (Strings, Tuples, Lists, etc).
Alphabet	A set of symbols used to build words of a certain language (e.g., the Latin alphabet for English, the Cyrillic alphabet for Russian, Kanji for Japanese, and so on). A program needs to be written in a recognizable script, such as Roman, Cyrillic, etc.

Lexis	A set of words the language offers its users (e.g., the word "computer" comes from the English language dictionary, while "cmoptrue" doesn't; the word "chat" is present both in English and French dictionaries, but their meanings are different). Each programming language has its dictionary and you need to master it; thankfully, it's much simpler and smaller than the dictionary of any natural language;
Syntax	A set of rules (formal or informal, written or felt intuitively) used to determine if a certain string of words forms a valid sentence (e.g., "I am a python" is a syntactically correct phrase, while "I a python am" isn't). Each language has its rules and they must be obeyed
Semantics	A set of rules determining if a certain phrase makes sense (e.g., "I ate a doughnut" makes sense, but "A doughnut ate me" doesn't). The program has to make sense.
Error Message	If the compiler finds an error, it finishes its work immediately. The only result in this case is an error message. The interpreter will inform you where the error is located and what caused it.
Module	Python's add-ons are called modules
Hierarchy of priorities	The phenomenon that causes some operators to act before others is known as the hierarchy of priorities.