**1.What is the difference between '==' and 'is' operator in Python? Provide an example for each.**

**A .** The '==' operator compares the value or equality of two objects, whereas the Python 'is' operator checks whether two variables point to the same object in memory location.

#equality operator

a = 3

b = 3

a == b

True

#return True

##For "is" operator,

print(id(a))

2813000247664

print(id(b))

2813000247664

a is b

True

# Return True because both a and b is pointing to the same object

# Here variable a is assigned to new variable c,

# which holds same object and same memory location

c=a

id(c)

2813000247664

a is c

True

**2.Explain the concept of mutability of objects in Python with suitable examples.**

**A.**  Concept of Mutability of objects in python -

In python objects like list, string, dictionary, set etc, if we can replace any item form the object then this object is mutable otherwise immutable.

example,

#list

l = [1,2,3,"shar","a",5+2j]

l[2] = 6

print(l)

[1,2,6,"shar","a",5+2j]

#list is mutable, it support item assignment.

#string

s = "sharmi"

s[3] = "x"

#error will occur because string doesnot support item assignment,

# string is immutable

we can alter the object's internal data or attributes without creating a new object.

**3.What are short-circuit operators in Python? Provide an example to demonstrate their usage.**

**A.** By short-circuiting, we mean the stoppage of execution of boolean operation if the truth value of expression has been determined already. In python, short-circuiting is supported by various boolean operators and functions.

# python code to demonstrate short circuiting using and and or

# function

def check():

    return "hello"

# using an expression to demonstrate

# prints "hello", and gets executed

# as both are required to check truth value

print (1 and check())

-> hello

# using an expression to demonstrate

# prints 1

# as only if 1st value is true, or

# doesnt require call check() func

print (1 or check())

-> 1

# using an expression to demonstrate

# prints "hello"

# the value returns true when check

# is encountered. 1 is not executed

print (0 or check() or 1)

\_> hello

# using an expression to demonstrate

# prints 1

# as last value is required to evaluate

# full expression because of "and"

print (0 or check() and 1)

-> 1

**4.How does the 'and' operator work in Python? Provide an example to illustrate its functionality.**

**A.**

The "and" operator is a logical operator it will return "True" if both operands are True. If not, it return False.

# here both operands are True, So, it will return 1

print(1 and 1)

-> 1

print(1 and 0)

-> 0

"and" operator check both operands are True or not.

a = 10

b = 2

c = -5

if a > 0 and b > 0:

print("the numbers are greater than zero")

if a > 0 and b > 0 and c > 0:

    print("The numbers are greater than 0")

else:

    print("Atleast one number is not greater than 0")

**5.Explain the ternary conditional operator in Python with an example.**

**A.** The ternary conditional operator in python is a shorter way of if-else statement.

# here the condition check if a>0, then execute.

# here it"s True as a =10then print "a is greater than 0" ,

#otherwise "a is not greater than 0"

a = 10

print("a is greater than 0" if a>0 else "a is not greater than 0")

-> a is greater than 0

**6.What is the difference between 'is' and 'is not' operators in Python? Provide an example for each.**

**A.** In python , "is" and "is not "operators compare the identity of objects.

In python, everything is an object and every object stores in a memory location.

* "is" operator checks whether both the variables points refer to same object in memory. If its true then it will return "True"
* "is not" operator checks whether both variable points refer to different object in memory location.

example,

here identity of a and b are same because the value of variables refer same object in memory location

a = 10

b = 10

print(a is b)

print(id(a), id(b))

-> True

140724791569616 140724791569616

here identity of a and b are different because list refers different object in memory location.

a = [1,2]

b = [1,2]

print(a is b)

print(id(a), id(b))

-> False

2196776121152 2196780432448

**7.Discuss the concept of shallow and deep copy in Python with respect to mutability of objects.**

**A.** Concept of shallow and deep copy in python with respect to object:

* for shallow copy, the change made in the copy list affect the original list, indicating the list is swallow copied.
* for the deep copy, the change made in the copy list did not affect the original list, indicating the list is deeply copied.

import copy

x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# create a new object that is a shallow copy of x.

y = list(x)

# create a new object that is a deep copy of x.

z = copy.deepcopy(x)

print("Original list x:", x)

print("shallow Copied list y:", y)

print("deep Copied list z:", z)

print(id(x), id(y), id(z))

-> Original list x: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

shallow Copied list y: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

deep Copied list z: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

1658515396480 1658515837760 1658515334016

# mutate items of y and z, for shallow copy y effects the original list x

# but for z deep copy,will not effect the original list

y[0].append(10)

z[1].append(5)

print("Original list x:", x)

print("shallow Copied list y:", y)

print("deep Copied list z:", z)

print(id(x), id(y), id(z))

-> Original list x: [[1, 2, 3, 10], [4, 5, 6], [7, 8, 9]]

shallow Copied list y: [[1, 2, 3, 10], [4, 5, 6], [7, 8, 9]]

deep Copied list z: [[1, 2, 3], [4, 5, 6,5], [7, 8, 9]]

1658515396480 1658515837760 1658515334016

**8.How can you modify a mutable object in a function without impacting the original object outside the function?**

**A.** Using deep copying, we can modify a mutable object in a function without impacting the original object outside the function. A deep copy creates a new object and recursively adds copies of the objects found in the original, effectively duplicating everything.

**9.What is the result of the expression '3 > 5 or 5 < 3' in Python? Explain the evaluation process.**

**A.**

3>5 or 5>3

# False or True

# 'OR' operator checks if the first value is True/False.

# here 1st value is (3>5) is False, then check the another (5>3),that's True,

# So, it will return True

result is = True

**10.Explain the purpose and usage of the 'not' operator in Python with an example.**

**A.** The "not" operator is a logical operator that is usually used to figure out the negation or opposite boolean value of the given operand.

x = True

print(not x)

-> False

**11.How does the 'in' operator work in Python? Provide an example to illustrate its usage.**

**A.** The "in" operator in python , it checks whether a value is exist in a sequence (like list, tuples, dictionary etc) then return boolean value : True if the value is in sequence otherwise False.

example,

a = [1,2,3,4]

print(3 in a)

-> True

**12.Discuss the concept of truthy and falsy values in Python with examples.**

**A.** A truthy value in python is a value which is evaluate to True in boolean context.

True

and a falsy value in python is a value that evaluates to False in boolean context.

False

The falsy values are empty seqences(list, tuple, dictionary etc), None, False, 0, 0.0 etc

a = 0

if a:

print(a)

-> no output

# value of a is zero, it's not True, So it will not pass the condition.

**13.Explain the concept of immutability of objects in Python. How is it different from mutability?**

**A.** Immutability - In python, Immutability of objects mean we can't changethe value of the object. If it is not possible to alter the value of an object in Python, it is known as an immutable object. Once an immutable object is created, it becomes permanent and unchangeable.

Mutability vs Immutability -

Mutable objects in Python are those that can be changed after they are created, like lists or dictionaries or sets. Immutable objects, on the other hand, cannot be changed after they are created, like strings, integers, tuples etc.

In mutable data types, we can modify the already existing values of the data types Or we may add new values or remove the existing values from our data types. Basically, we may perform any operation with our data without having to create a new copy of our data type. Hence, the value assigned to any variable can be changed. But an immutable data types is not changeable.