Name: K. Praveen kumar

Python assignment-4 (15-12-2023)

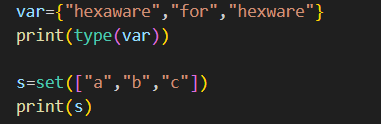
**Set:**

A set in python is an unordered collection data type that is iterable, mutable and has no duplicate values.

Sets are represented by { }.

Since sets are unordered, we cannot access items using indexes as we do in the lists.

Example:



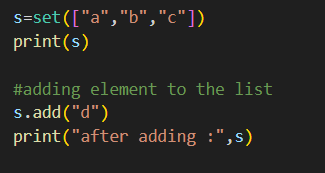
Output:



**Type Casting with python set method:**

Set() method is used for type casting.

#type casting list to set

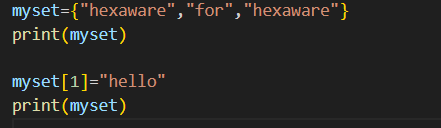


Output:

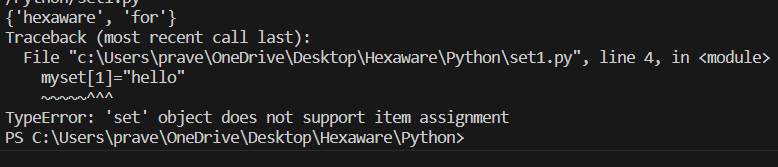


**Check unique and Immutable with python set:**

Python sets cannot have a duplicate value and once it is created we cannot change its value.



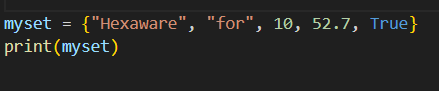
**Output:**



Set is immutable so the values cannot be changed.

**Heterogeneous Elements with Python set:**

Set can store a mixture of string, Integer, Boolean, etc…

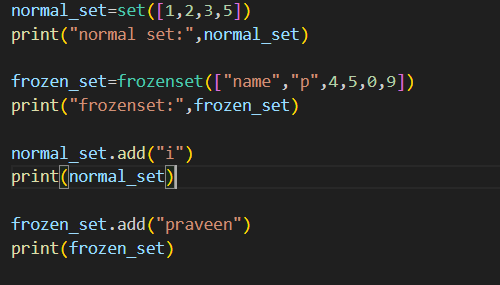


Output:

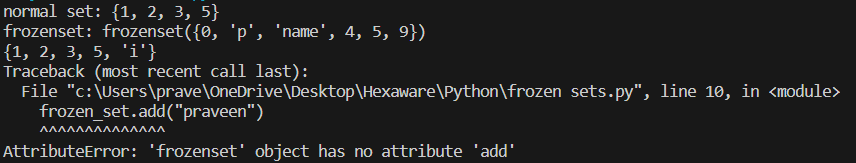


**Frozen Sets:**

Frozen sets in python are immutable objects that only support methods and operators that produce a result without affecting frozen sets



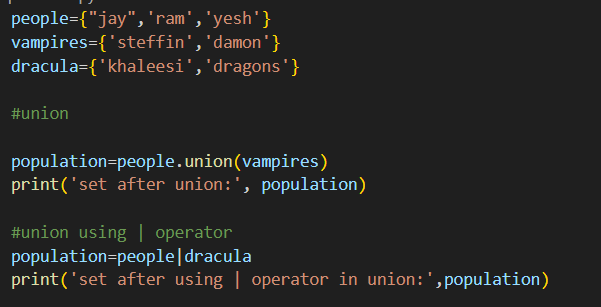
**Output:**



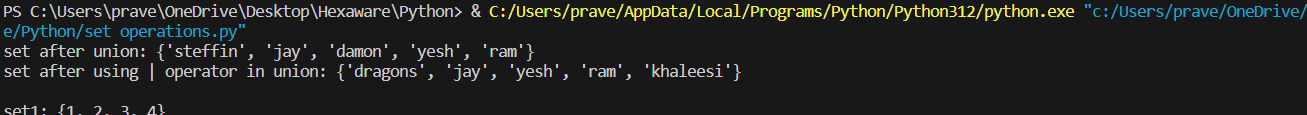
**Operation on sets:**

**Union operation:**

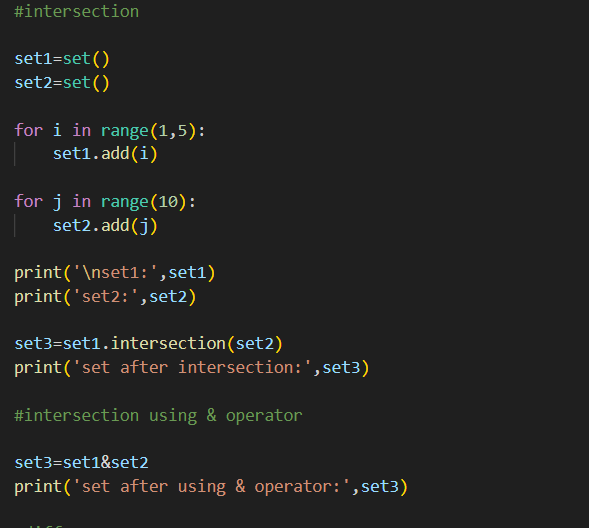
Two sets can be merged using union() function or ‘|’ (or) operator. Both hash table values are accessed and traversed with merge operation perform on them to contain the elements.



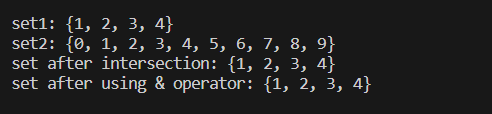
Output:



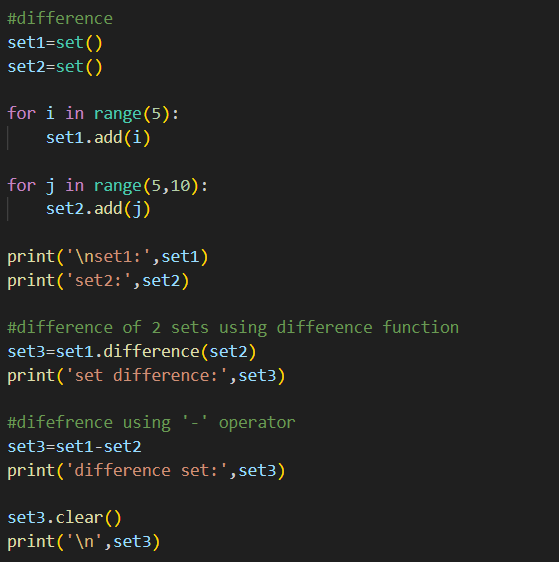
**Intersection:**This can be done through intersection() and ‘&’ operator. Common elements are selected .They are similar to iteration over the hash lists and combining the same values on both the tables.



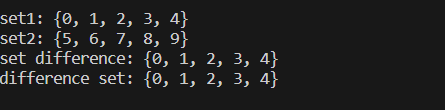
**Output:**



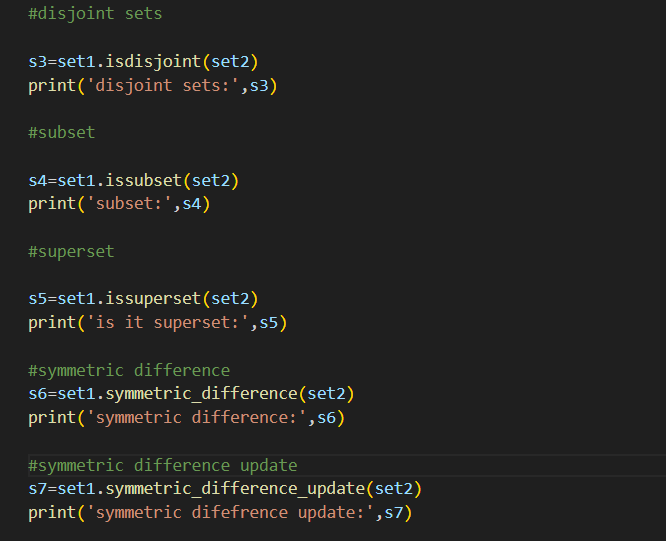
**Difference of sets:**



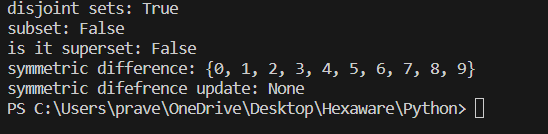
Output:



There are other operations like subset(), superset(), disjoint set(), symmetric\_difference() and symmetric\_difference\_update.



Output:



**Set Methods:**

**1.Adding elements to the set:**

#insertion

people={'john','snow','brad'}

print('people:',end=' ')

print(people)

people.add("daxit")

print(people)

#insertion using iterator

for i in range(1,6):

    people.add(i)

print('after adding elements:',people)

we can add any python iterable to a set using python add or python update function.

s={'g','e','e','k','s'}

t=('f','o')

l=['a','e']

#add tuple to set

s.add(t)

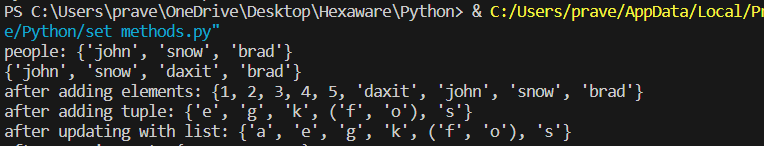
print('after adding tuple:',s)

#add list to set

s.update(l)

print('after updating with list:',s)

**Output:**

****

**2.Copy():**

The copy() method returns a shallow copy of the set in python. If we use “=” to copy a set to another set. When we modify in the copies set , the changes are also reflected in the original set.

#copy

s1={1,2,3,4}

s2=s1.copy()

print(s2)



**3.Discard():**

Discard() is a built-in method to remove elements from the set. It takes only one argument.

#discard

s={1,2,3,4,5,6,7}

s.discard(3)

print(s)

myset={'a',1,"pyt",2,'b',"hexaware",8}

myset.discard("hexaware")

print('after discard:',myset)

output:



The other methods are like remove(), pop() and clear().

s={'h','e','x','a','w','a','r','e'}

print("\noriginal set:",s)

s.add('c')

print('after insertion:',s)

s.discard('x')

print('after discarding',s)

s.remove('e')

print('after removing',s)

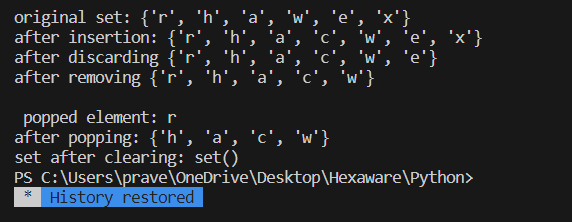
print("\n popped element:",s.pop())

print('after popping:',s)

s.clear()

print("set after clearing:",s)

output:



Python Functions:

Function is a block of statements that return the specific task.

Benefits:

1.Increase Code readability

2.Increase Code Reusability

Syntax:

Def function\_name(parameters):

# statement(s)

return expression

Types of function:

1. Built-in library Functions: These are standard functions in python that are available to use.
2. User defined Functions: we can create our own functions based on our requirements.

**Creating a Function:**

def fun():

print(“welcome to Python”)

**calling a function:**

def fun():

print(“welcome to python”)

fun()

**Functions():**

def fun():

    print("welcome to python")

fun()

def fun(n1,n2):

    n3=n1+n2

    print(n3)

fun(4,5)

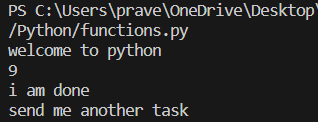
def printsuccess():

    print('i am done')

    print('send me another task')

printsuccess()

**output:**

****

**Function with Parameters:**

#function with parameters

def add(num1: int, num2: int) -> int:

    """Add two numbers"""

    num3 = num1 + num2

    return num3

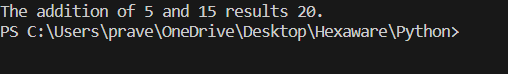
# Driver code

num1, num2 = 5, 15

ans = add(num1, num2)

print(f"The addition of {num1} and {num2} results {ans}.")

**output:**

****

**Example 2 : To print prime numbers**

def is\_prime(n):

   if n in [2, 3]:

       return True

   if (n == 1) or (n % 2 == 0):

       return False

   r = 10

   while r \* r <= n:

       if n % r == 0:

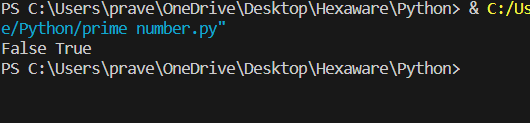
           return False

       r += 2

   return True

print(is\_prime(78), is\_prime(79))

**output:**

****

**Python Function Arguments:**

Arguments are the values passed inside the paranthesis of the function.

Example:

def evenodd(x):

if(x%2==0):

print(“even”)

else:

print(“odd”)

evenodd(2)

evenodd(3)

**output:**

even

odd

**Types :**

1.Default Arguments: It is a parameter that assumes a default value if a value is not provided in the function call for that argument.

#default arguments

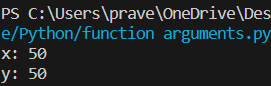
def myfun(x,y=50):

    print("x:",y)

    print("y:",y)

myfun(10)

Output:



**2.Keyword Arguments:** This allows the caller to specify the argument name with values so that the caller doesn’t need to remember the order of parameters.

#keyword arguments

def student(firstname,lastname):

    print(firstname,lastname)

student(firstname='hexa',lastname='practice')

student(lastname='practice',firstname='hexa')

output:



**3.Positional Arguments:** we use this during the function call so that the first argument is assigned to name and the second argument is assigned to age. If you change the order, the output will be like unordered and not meaningful.

#positional arguments

def nameAge(name,age):

    print("hi,I am",name)

    print("my age is",age)

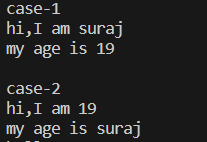
print("case-1")

nameAge("suraj",19)

print("\ncase-2")

nameAge(19,"suraj")

output:



**4.Arbitrary Keyword Arguments:**

\*args and \*\*kwargs can pass a variable number of arguments to a function using special symbols.

1. \*args in python (non-keyword arguments)

2. \*\*kwargs in python (keyword arguments)

1.non-keyword arguments:

#non-keyword arguments

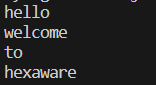
def myfun(\*argv):

    for arg in argv:

        print(arg)

myfun('hello','welcome','to','hexaware')

output:



2.keyword arguments:

#keyword arguments

def myFun(\*\*kwargs):

    for key, value in kwargs.items():

        print("%s == %s" % (key, value))

myFun(first='Hexa', mid='for', last='Hexa')

output:

