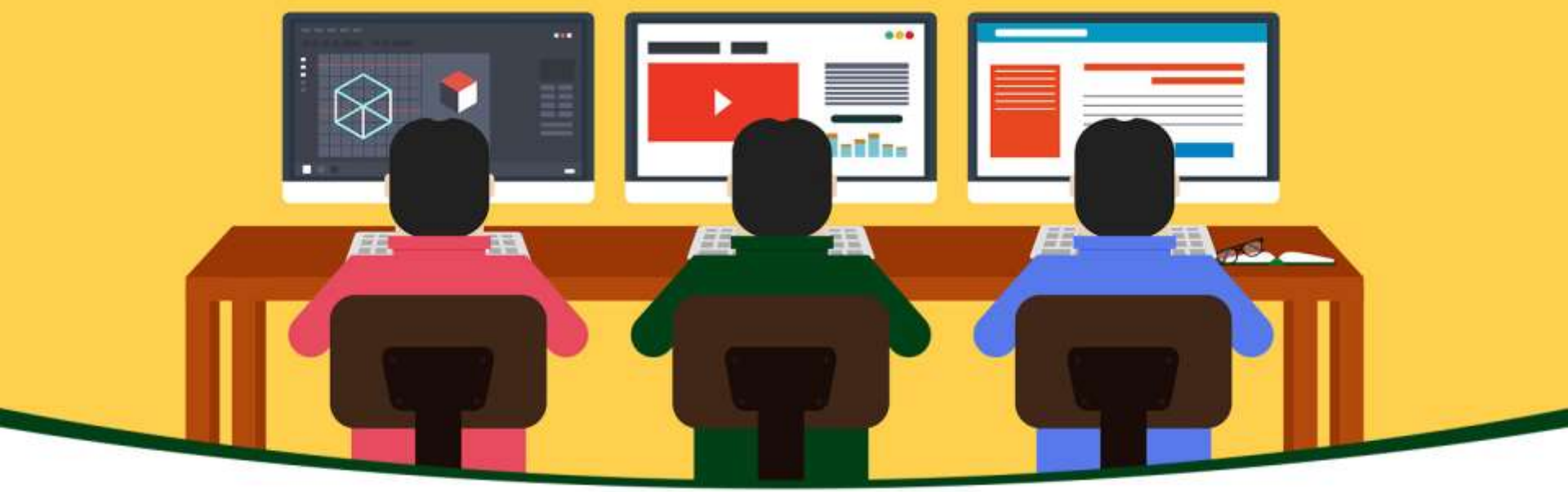

OPEN ELECTIVE
OCS1903 - PROGRAMMING USING PYTHON
WEEK 8 - SET

CONTENTS

- Set
- Set Creation
- Set Methods
- Set Operations
- Frozenset

Set-{}

- A Python set is the collection of the unordered heterogenous items
- Set elements are unique. Duplicate elements are not allowed.
- A set is mutable, but the elements contained in the set must be of an immutable type.
- There is no index attached to the elements of the set.

Eg:{2,4,1,7,8,9}

Set Creation

- The set can be created by enclosing the comma-separated immutable items with the curly braces {}.
- Python also provides the set() method, which can be used to create the set by the passed sequence.

```
>>> set1={"hi",1,2,3.14}
>>> set1
{1, 2, 3.14, 'hi'}
>>> set1={[1,2,3],(1,2,3)}
Traceback (most recent call last):
  File "<pyshell#152>", line 1, in <module>
    set1={[1,2,3],(1,2,3)}
TypeError: unhashable type: 'list'
```

```
>>> set1=set([3,4,2,3,1,5,2])
>>> set1
{1, 2, 3, 4, 5}
```

This Cleary shows that a set contains unordered and unique elements

Built-in method-Adding Elements

The add() method is used to add a single element to the set.

```
>>> set1
{1, 2, 3, 4, 5}
>>> set1.add(6)
>>> set1.add(5)
>>> set1
{1, 2, 3, 4, 5, 6}
```

```
>>> set1
{1, 8, 9, -2, -1}
>>> set1.update([6,6,5,4,3])
>>> set1
{1, 3, 4, 5, 6, 8, 9, -2, -1}
```

The update() method is used to add multiple elements to the set

Built-in method-Deleting Elements

The discard method is used to remove an item from the set.

```
>>> set1
{1, 3, 4, 5, 6, 9, -2, -1}
>>> set1.discard(5)
>>> set1
{1, 3, 4, 6, 9, -2, -1}
>>> set1.discard(11)
>>> set1
{1, 3, 4, 6, 9, -2, -1}
```

Deleting Element

```
>>> set1
{1, 3, 4, 6, 9, -2, -1}
>>> set1.remove(1)
>>> set1
{3, 4, 6, 9, -2, -1}
>>> set1.remove(1)
Traceback (most recent call last):
  File "<pyshell#195>", line 1, in <module>
    set1.remove(1)
KeyError: 1
```

The remove method is used to remove an item from the set

Python Set Operations

Operation	Equivalent	Result
`len(s)`		number of elements in set *s* (cardinality)
`x in s`		test *x* for membership in *s*
`x not in s`		test *x* for non-membership in *s*
`s.issubset(t)`	`s <= t`	test whether every element in *s* is in *t*
`s.issuperset(t)`	`s >= t`	test whether every element in *t* is in *s*
`s.union(t)`	`s | t`	new set with elements from both *s* and *t*
`s.intersection(t)`	`s & t`	new set with elements common to *s* and *t*
`s.difference(t)`	`s - t`	new set with elements in *s* but not in *t*
`s.symmetric_difference(t)`	`s ^ t`	new set with elements in either *s* or *t* but not both
`s.copy()`		new set with a shallow copy of *s*

Python Set Operations

Size membership and identity

```
>>> set1
{3, 4, 6, 9, -2, -1}
>>> len(set1)
6
>>> 3 in set1
True
>>> set2={3,9,4,6,-2,-1}
>>> set2
{3, 4, 6, -2, 9, -1}
>>> set1 is set2
False
```

Superset and subset

```
>>> s={3,4,2,1}
>>> t={1,2,3,4,5,6}
>>> s
{1, 2, 3, 4}
>>> t
{1, 2, 3, 4, 5, 6}
>>> s>=t
False
>>> s.issuperset(t)
False
>>> s<=t
True
>>> s.issubset(t)
True
```

Union and Intersection

```
>>> set1=s|t
>>> set1
{1, 2, 3, 4, 5, 6}
>>> set2=s&t
>>> set2
{1, 2, 3, 4}
```

Python Set Operations

Difference and Symm~ Difference

```
>>> s
{1, 2, 3, 4}
>>> t
{1, 2, 3, 4, 5, 6}
>>> s-t
set()
>>> t-s
{5, 6}
>>> s={1,2,4}
>>> t={4,3,7}
>>> s^t
{1, 2, 3, 7}
```

Copy()

```
>>> m=s.copy()
>>> m
{1, 2, 4}
>>> m.add(3)
>>> m
{1, 2, 3, 4}
>>> s
{1, 2, 4}
```

Frozenset

- The frozen sets are the immutable form of the normal sets.
- The elements of the frozen set cannot be changed after the creation.
- We cannot change or append the content of the frozen sets by using the methods like add() or remove().
- The frozenset() method is used to create the frozenset object.
- The iterable sequence is passed into this method which is converted into the frozen set as a return type of the method.

Frozenset

```
>>> fro1=frozenset([1,2,3,8,8,5])
>>> fro1
frozenset({1, 2, 3, 5, 8})
>>> fro1.add(9)
Traceback (most recent call last):
  File "<pyshell#234>", line 1, in <module>
    fro1.add(9)
AttributeError: 'frozenset' object has no attribute 'add'
```

Example 1

Write a Python program that takes a complete sentence as an input and remove duplicate word in it and then counts all the words which have a length greater than 3.

Input:
"we are good are we good"
Output:
we are good
Count=1

```
s=input("Enter the sentence")
l1=s.split()

set1=set(l1)
l1=list(set1)
print("After removing duplicates:",l1)
count=0
for i in l1:
    if(len(i)>3):
        count=count+1
print(count)
```

Example 2

Write a Python program to return a new set with unique items from both sets by removing duplicates.

Input:
set1 = {10, 20, 30, 40, 50}
set2 = {30, 40, 50, 60, 70}
Output:
{70, 40, 10, 50, 20, 60, 30}

```
set1 = {10, 20, 30, 40, 50}
set2 = {30, 40, 50, 60, 70}

print(set1.union(set2))
```

Example 3

Write a Python program to create an intersection of sets.

Program:

```
setx = set(["green", "blue"])
sety = set(["blue", "yellow"])
print("Original set elements:")
print(setx) print(sety)
print("\nIntersection of two said sets:")
setz = setx & sety print(setz)
```

Output:

```
Original set elements:
{'green', 'blue'}
{'blue', 'yellow'}

Intersection of two said sets:
{'blue'}
```

Example 4

Write a Python program to use of frozensets.

Note: Frozensets behave just like sets except they are immutable.

Program:

```
x = frozenset([1, 2, 3, 4, 5])
y = frozenset([3, 4, 5, 6, 7])
print(x.isdisjoint(y))
print(x.difference(y))
print(x | y)
```

Output:

```
False
frozenset({1, 2})
frozenset({1, 2, 3, 4, 5, 6, 7}
```


THANK
YOU