# BCSE101E

# Computer Programming: Python

*Course Faculty: Rajesh M – SCOPE, VIT Chennai*

## *Module – 4 Collections in Python (List, Tuple, Dictionary, Set)*

## Sets in Python

## Problem:

An University has published the results of the term end examination conducted in April. List of failures in physics, mathematics, chemistry and computer science is available. Write a program to find the number of failures in the examination. This includes the count of failures in one or more subjects

## PAC For University Result Problem

| Input | Processing | Output |
|---|---|---|
| Read the register number of failures in Maths, Physics, Chemistry and Computer Science | Create a list of register numbers who have failed in one or more subjects<br>Count the count of failures | Print Count |

## Pseudocode

```
READ maths_failure, physics_failure, chemistry_failure and cs_failure
Let failure be empty
FOR each item in maths_failure
        ADD item to failure
FOR each item in physics_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
FOR each item in chemistry_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
FOR each item in cs_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
SET count = 0
FOR each item in failure
        count = count + 1
PRINT count
```

## Sets

- an unordered collection of unique and immutable objects that supports operations corresponding to mathematical set theory

- Set is **mutable**

- **No duplicates**

- Sets are **iterable**, can **grow and shrink** on demand, and may contain a variety of object types

- **Does not support indexing.**

```
x = {1, 2, 3, 4}

y = {'apple','ball','cat'}

x1 = set('spam')  # Prepare set from a string
print (x1)

{'s', 'a', 'p', 'm'}

x1.add('alot')    # Add an element to the set
print (x1)

{'s', 'a', 'p', 'alot', 'm'}
```

## Set Operations

Let S1 = {1, 2, 3, 4}

**Union (|)**

```
S2 = {1, 5, 3, 6} | S1
print(S2)                    # prints {1, 2, 3, 4, 5, 6}
```

**Intersection (&)**

```
S2 = S1 & {1, 3}
print(S2) # prints {1, 3}
```

## Difference (-)

```
S2 = S1 - {1, 3, 4}

print(S2)    # prints {2}
```

## Super set (>)

```
S2 = S1 > {1, 3}

print(S2)    # prints True
```

Empty sets must be created with the set built-in, and print the same way

```
S2 = S1 - {1, 2, 3, 4}
print(S2) # prints set() – Empty set
```

- **Empty curly braces represent empty dictionary but not set**

In interactive mode – type({}) gives

```
<class 'dict'>
```

```
>>> {1, 2, 3} | {3, 4}
{1, 2, 3, 4}

>>> {1, 2, 3} | [3, 4]
TypeError: unsupported operand type(s) for |: 'set' and 'list'

>>> {1, 2, 3} | set([3, 4]) #Convert list to set and work
{1,2,3,4}

>>> {1, 2, 3}.union([3, 4])
{1,2,3,4}

>>> {1, 2, 3}.union({3, 4})
{1,2,3,4}
```

## Immutable constraints and frozen sets

- Can **only** contain **immutable** (a.k.a. "hashable") **object types**
- **lists and dictionaries cannot be embedded in sets, but tuples can** if you need to store compound values.
- Tuples **compare by their full values** when used in set operations:

```
>>> S
{1.23}

>>> S.add([1, 2, 3])
TypeError: unhashable type: 'list'
```

```
>>> S.add({'a':1})
TypeError: unhashable type: 'dict'
```

**Works for tuples:**

```
>>> S.add((1, 2, 3))
>>> S
{1.23, (1, 2, 3)}

>>> S | {(4, 5, 6), (1, 2, 3)}
{1.23, (4, 5, 6), (1, 2, 3)}

>>> (1, 2, 3) in S # Check for tuple as a whole
True

>>> (1, 4, 3) in S
False
```

## clear()

**All elements will be removed** from a set.

```
>>> cities = {"Stuttgart", "Konstanz", "Freiburg"}
>>> cities.clear()
>>> cities
set() # empty
>>>
```

## Copy

Creates a **shallow copy**, which is returned.

```
>>> more_cities = {"Winterthur","Schaffhausen","St. Gallen"}
>>> cities_backup = more_cities.copy()

>>> more_cities.clear()

>>> cities_backup  # copied value is still available
{'St. Gallen', 'Winterthur',  'Schaffhausen'}
```

Just in case, you might think, an **assignment** might be enough:

```
>>> more_cities = {"Winterthur","Schaffhausen","St. Gallen"}
>>> cities_backup = more_cities #creates alias name

>>> more_cities.clear()
>>> cities_backup
set()
>>>
```

❑ **The assignment "cities_backup = more_cities" just creates a pointer, i.e. another name, to the same data structure.**

## difference_update()

**removes all elements of another set** from this set.
x.difference_update() is the same as "x = x - y"

```
>>> x = {"a","b","c","d","e"}
>>> y = {"b","c"}
>>> x.difference_update(y)
>>> x
{'a', 'e', 'd'}
```

## discard(el)

el will be removed from the set, if it is contained in the set and nothing will be done otherwise

```
>>> x = {"a","b","c","d","e"}
>>> x.discard("a")

>>> x
{'c', 'b', 'e', 'd'}

>>> x.discard("z")

>>> x
{'c', 'b', 'e', 'd'}
```

## remove(el)

- works like discard(), but if el is **not a member** of the set, a **KeyError** will be raised.

```
>>> x = {"a","b","c","d","e"}
>>> x.remove("a")
>>> x
{'c', 'b', 'e', 'd'}

>>> x.remove("z")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'z'
```

**isdisjoint()**

This method **returns True** if **two sets** have a **null intersection**

**issubset()**

x.issubset(y) **returns True**, if **x is a subset of y**

**"<="** is an abbreviation for **"Subset of"** and **">="** for "**superset of"**

**"<"** **i**s used to check if a set is a **proper subset** of a set

## issuperset()

x.issuperset(y) returns True, if x is a superset of y. ">=" - abbreviation for "issuperset of"

">" - to check if a set is a proper superset of a set

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d"}
>>> x.issuperset(y)
True
```

```
>>> x > y
True
>>> x >= y
True
>>> x >= x
True
>>> x > x
False
>>> x.issuperset(x)
True
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d"}
>>> x.issubset(y)
False
>>> y.issubset(x)
True
>>> x < y
False
>>> y < x # y is a proper  subset  of x
True
>>> x < x # a set is not a proper subset of oneself.
False
>>> x <= x
True
```

## pop()

- pop() **removes and returns an arbitrary set element.**

The method raises a **KeyError if the set is empty**

```
>>> x = {"a","b","c","d","e"}
>>> x.pop()
'a'
>>> x.pop()
'c'
```

- Sets themselves are **mutable** too, and so **cannot be nested in other sets directly**;
- if you need to store a set inside another set, the **frozenset** built-in call works just like set but creates an immutable set that cannot change and thus can be embedded in other sets.

## To create frozenset:

```
cities = frozenset(["Frankfurt", "Basel","Freiburg"])

cities.add("Strasbourg")          #cannot modify

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

## Set comprehensions

run a loop and collect the result of an expression on each iteration

result is a new set you create by running the code, with all the normal set behavior

```
>>> {x ** 2 for x in [1, 2, 3, 4]}
{16, 1, 4, 9}

>>> {x for x in 'spam'}
{'m', 's', 'p', 'a'}
```

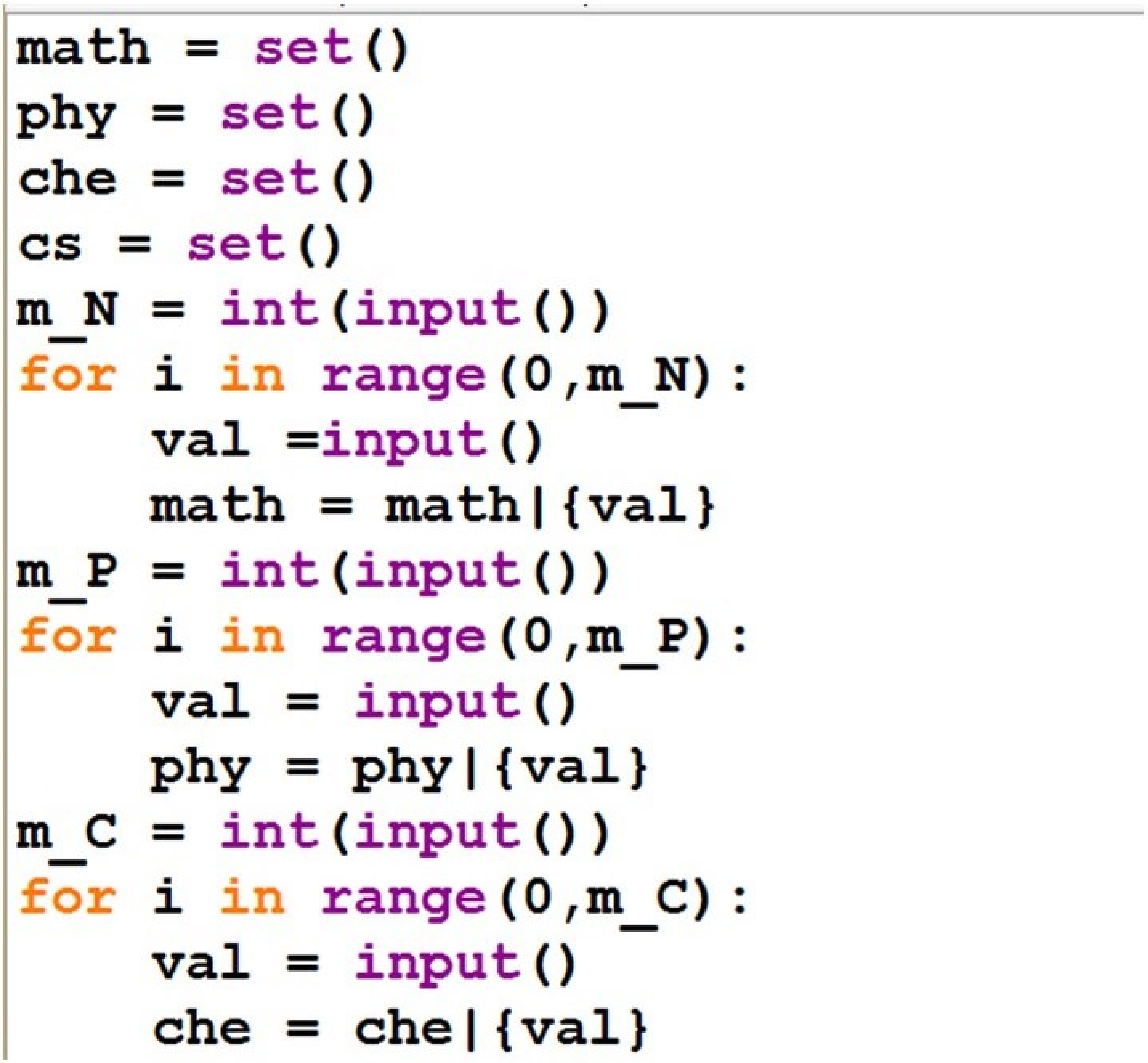
```
>>> S = {c * 4 for c in 'spam'}

>>> print(S)
{'pppp','aaaa','ssss', 'mmmm'}

>>>> S = {c * 4 for c in 'spamham'}
{'pppp','aaaa','ssss', 'mmmm','hhhh'}

>>>S | {'mmmm', 'xxxx'}
{'pppp', 'xxxx', 'mmmm', 'aaaa', 'ssss'}

>>> S & {'mmmm', 'xxxx'}
{'mmmm'}
```



```
math = set()
phy = set()
che = set()
cs = set()
m_N = int(input())
for i in range(0,m_N):
    val =input()
    math = math|{val}
m_P = int(input())
for i in range(0,m_P):
    val = input()
    phy = phy|{val}
m_C = int(input())
for i in range(0,m_C):
    val = input()
    che = che|{val}
```

```
m_CS = int(input())
for i in range(0,m_CS):
    val = input()
    cs = cs|{val}
failure = math|phy|che|cs
print(len(failure))
```

## Problem

A hospital has received a set of lab reports. Totally five tests are conducted in the lab and the report is prepared in such a way that the 'n^th' number correspond to value of $test_n$. If the test was not performed for the particular patient then 'N' is written in the corresponding place. Write a program to print if the test result is normal or not normal by using the values in Table 1. Since the value is sensitive, provide a mechanism so that the values do not get altered.

| Name of the Test | Minimum Value | Maximum Value |
|---|---|---|
| Test1 | 20 | 30 |
| Test2 | 35.5 | 40 |
| Test3 | 12 | 15 |
| Test4 | 120 | 150 |
| Test5 | 80 | 120 |

```
READ lab_report
FOR i =0 to 5
    IF lab_report[i] != N
        IF lab_report[i] < min of $test_i$ OR
            lab_report[i] >max of $test_i$ THEN
            PRINT 'abnormal'
        ELSE
            PRINT 'normal'
        END IF
    END IF
END FOR
```

## Exercise 1

While John got ready for the office a shopping list was given by his son and daughter. While returning home John decides to buy the items from a shop. He decides to buy the items common in both the list, then the items listed only by his son and at last the items listed only by his daughter. Write a program to help him in achieving the task.

## Exercise 2

A marketing company has branch in a set of cities 'S'. The company sends three of their sales man to various cities in set 'S'. In the middle of the year, help the manager to find out the cities that are already visited by the sales men and the cities that are yet to be visited.

Thank You!