Lab Manual # 01

FBDA

Department of Computer Science
FAST-NU, Lahore, Pakistan

Lab Protocols:

1. Carefully read and follow all instructions
2. No evaluation would be done after Lab's timing. So, keep the track of time.
3. Do keep in mind that sharing the code, discussing it during lab or looking for online solution is highly unethical, and all actions would be considered as plagiarism.
4. Plagiarism will result in serious penalty

Objective : Intro to Python

In computer programming, loops are used to repeat a block of code.

For example, if we want to show a message 100 times, then we can use a loop. It's just a simple example; you can achieve much more with loops.

There are 2 types of loops in Python:

- for loop
- while loop

Python for Loop

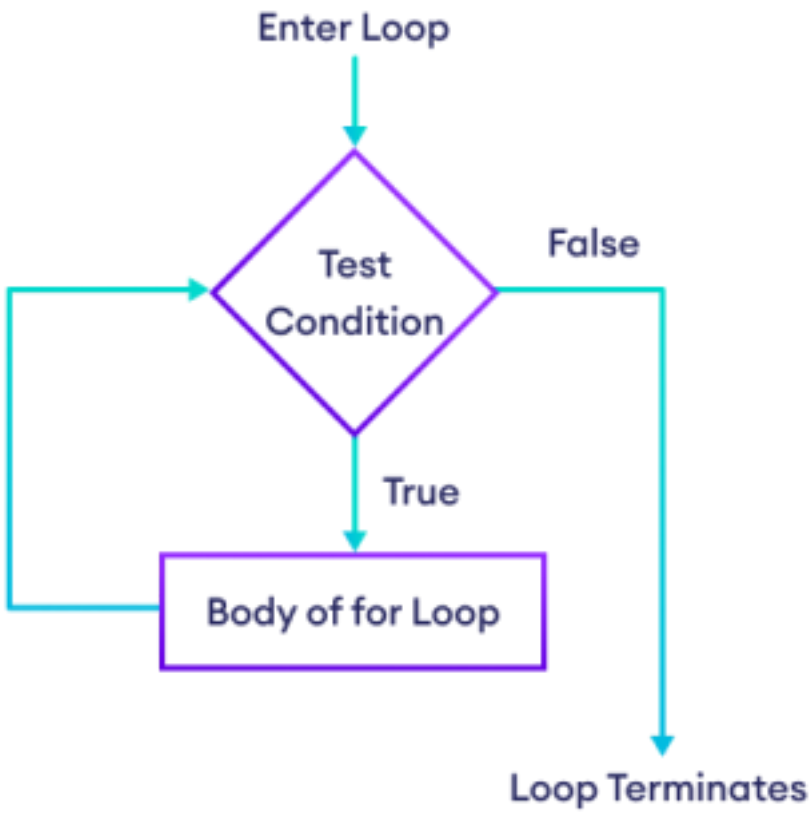
In Python, the `for` loop is used to run a block of code for a certain number of times. It is used to iterate over any sequences such as list, tuple, string, etc.

The syntax of the `for` loop is:

```
for val in sequence:
    # statement(s)
```

Here, `val` accesses each item of sequence on each iteration. Loop continues until we reach the last item in the sequence.

Flowchart of Python for Loop



Working of Python for loop

Example: Loop Over Python List

```
languages = ['Swift', 'Python', 'Go', 'JavaScript']

# access items of a list using for loop
for language in languages:
    print(language)
```

Output

```
Swift
Python
Go
JavaScript
```

In the above example, we have created a list called `languages`.

Initially, the value of `language` is set to the first element of the array,i.e. `Swift`, so the print statement inside the loop is executed.

`language` is updated with the next element of the array and the print statement is executed again. This way the loop runs until the last element of an array is accessed.

Python for Loop with Python range()

A range is a series of values between two numeric intervals.

We use Python's built-in function `range()` to define a range of values.
For example,

```
values = range(4)
```

Here, 4 inside `range()` defines a range containing values 0, 1, 2, 3.

In Python, we can use `for` loop to iterate over a range. For

example,

```
# use of range() to define a range of values
values = range(4)

# iterate from i = 0 to i = 3
for i in values:
    print(i)
```

Output

```
0
1
2
3
```

In the above example, we have used the `for` loop to iterate over a range from 0 to 3.

The value of `i` is set to 0 and it is updated to the next number of the range on each iteration. This process continues until 3 is reached.

Iteration	Condition	Action
1st	`True`	`0` is printed. `i` is increased to 1.
2nd	`True`	`1` is printed. `i`

is increased to 2. 3rd `True` `2` is printed. `i` is increased to 3. 4th `True`

`3` is printed. `i` is increased to 4.

5th `False` The loop is terminated

Note: To learn more about the use of `for` loop with range, visit Python range().

Python for loop with else

A `for` loop can have an optional `else` block as well. The `else` part is executed when the loop is finished. For example,

```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

Output

```
0
1
```

```
5
No items left.
```

Here, the `for` loop prints all the items of the `digits` list. When the loop finishes, it executes the `else` block and prints `No items left.`

Note: The else block will not execute if the for loop is stopped by a break statement.

In computer programming, we use the `if` statement to run a block code only when a certain condition is met.

For example, assigning grades (A, B, C) based on marks obtained by a student.

1. if the percentage is above 90, assign grade A
2. if the percentage is above 75, assign grade B
3. if the percentage is above 65, assign grade C

In Python, there are three forms of the `if...else` statement.

1. `if` statement
2. `if...else` statement
3. `if...elif...else` statement

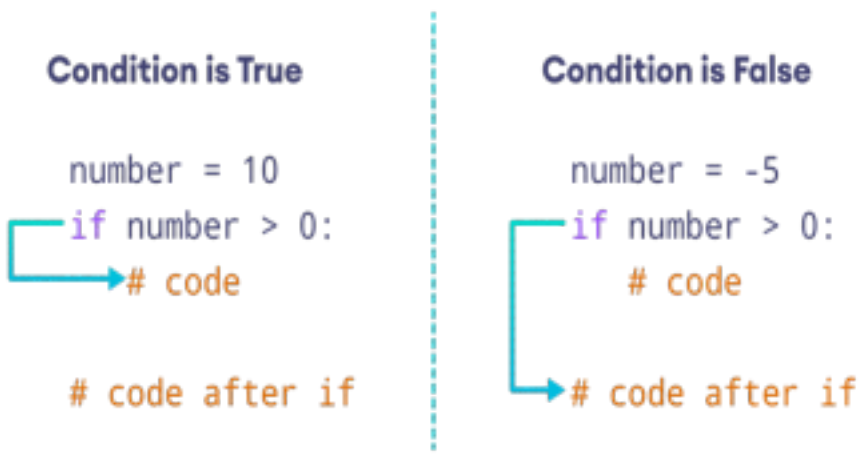
1. Python if statement

The syntax of `if` statement in Python is:

```
if condition:
    # body of if statement
```

The `if` statement evaluates `condition`.

1. If `condition` is evaluated to `True`, the code inside the body of `if` is executed.
2. If `condition` is evaluated to `False`, the code inside the body of `if` is skipped.



Working of if Statement

Example 1: Python if Statement

```
number = 10

# check if number is greater than 0
if number > 0:
    print('Number is positive.')

print('The if statement is easy')
```

Output

```
Number is positive.
The if statement is easy
```

In the above example, we have created a variable named `number`. Notice the test condition,

```
number > 0
```

Here, since `number` is greater than 0, the condition evaluates `True`. If we change the value of variable to a negative integer. Let's say -5.

```
number = -5
```

Now, when we run the program, the output will be:

```
The if statement is easy
```

This is because the value of `number` is less than 0. Hence, the condition evaluates to `False`. And, the body of `if` block is skipped.

2. Python if...else Statement

An `if` statement can have an optional `else` clause.

The syntax of `if...else` statement is:

```
if condition:
    # block of code if condition is True
else:
    # block of code if condition is False
```

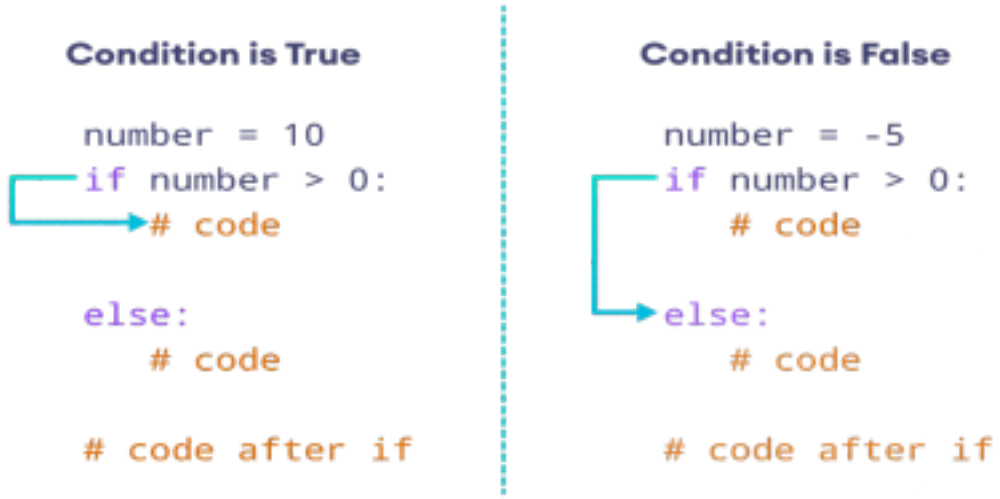
The `if...else` statement evaluates the given `condition`:

If the condition evaluates to `True`,

- the code inside `if` is executed
- the code inside `else` is skipped

If the condition evaluates to `False`,

- the code inside `else` is executed
- the code inside `if` is skipped



Working of if...else Statement

Example 2. Python if...else Statement

```
number = 10

if number > 0:
    print('Positive number')

else:
    print('Negative number')

print('This statement is always executed')
```

Output

```
Positive number
This statement is always executed
```

In the above example, we have created a variable named `number`. Notice the test condition,

```
number > 0
```

Since the value of `number` is 10, the test condition evaluates to `True`. Hence code inside the body of `if` is executed.

If we change the value of variable to a negative integer. Let's say -5.

```
number = -5
```

Now if we run the program, the output will be:

```
Number is negative.
This statement is always executed.
```

Here, the test condition evaluates to `False`. Hence code inside the body of `else` is executed.

3. Python if...elif...else Statement

The `if...else` statement is used to execute a block of code among two alternatives.

However, if we need to make a choice between more than two alternatives, then we use the `if...elif...else` statement.

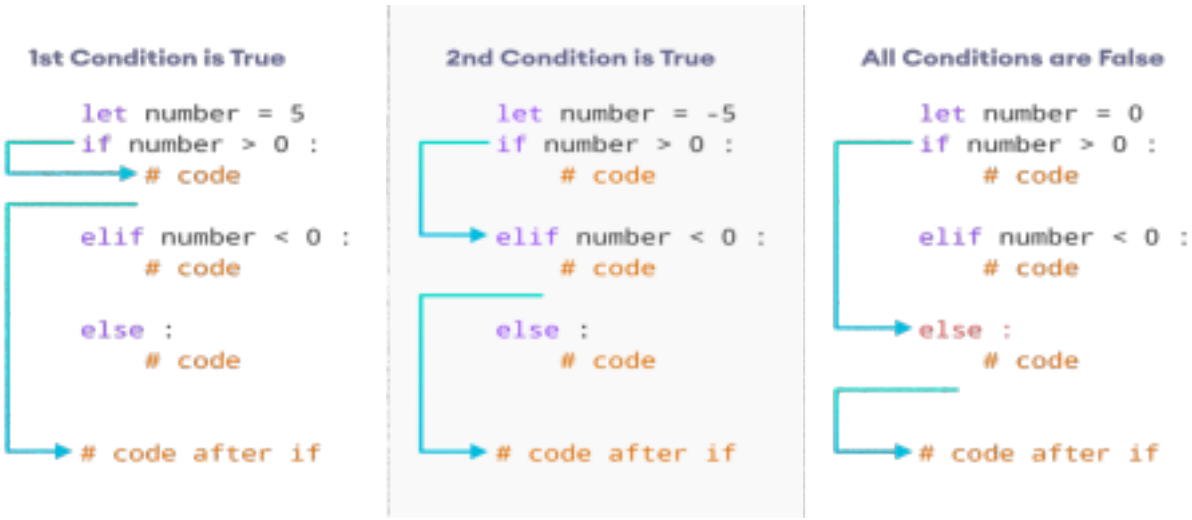
The syntax of the `if...elif...else` statement is:

```
if condition1:
```

```
    # code block 1

elif condition2:
    # code block 2

else:
    # code block 3
```

Here,

1. If condition1 evaluates to `true`, code block 1 is executed. 2. If condition1 evaluates to `false`, then condition2 is evaluated. 1. If condition2 is `true`, code block 2 is executed.
2. If condition2 is `false`, code block 3 is executed.



Working of if...elif Statement

Example 3: Python if...elif...else Statement

```
number = 0

if number > 0:
    print("Positive number")
```

```
elif number == 0:
    print('Zero')
else:
    print('Negative number')

print('This statement is always executed')
```

Output

```
Zero
This statement is always executed
```

In the above example, we have created a variable named `number` with the value 0. Here, we have two condition expressions:

Here, both the conditions evaluate to `False`. Hence the statement inside the body of `else` is executed.

Python Nested if statements

We can also use an `if` statement inside of an `if` statement. This is known as a nested if statement.

The syntax of nested if statement is:

```
# outer if statement
if condition1:
    # statement(s)
```

```
    # inner if statement
    if condition2:
        # statement(s)
```

Notes:

- We can add `else` and `elif` statements to the inner `if` statement as required.
- We can also insert inner `if` statement inside the outer `else` or `elif` statements(if they exist)
- We can nest multiple layers of `if` statements.

Example 4: Python Nested if Statement

```
number = 5

# outer if statement
if (number >= 0):
    # inner if statement
    if number == 0:
      print('Number is 0')

    # inner else statement
    else:
        print('Number is positive')

# outer else statement
else:
    print('Number is negative')
```

```
# Output: Number is positive
```

In the above example, we have used a nested if statement to check whether the given number is positive, negative, or 0.

A function is a block of code that performs a specific task.

Suppose, you need to create a program to create a circle and color it. You can create two functions to solve this problem:

- create a circle function
- create a color function

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

Types of function

There are two types of function in Python programming:

- Standard library functions - These are built-in functions in Python that are available to use.
- User-defined functions - We can create our own functions based on our

requirements.

Python Function Declaration

The syntax to declare a function is:

```
def function_name(arguments):
    # function body

    return
```

Here,

- `def` - keyword used to declare a function
- `function_name` - any name given to the function
- `arguments` - any value passed to function
- `return` (optional) - returns value from a function

Let's see an example,

```
def greet():
    print('Hello World!')
```

Here, we have created a function named `greet()`. It simply prints the text `Hello World!`.

This function doesn't have any arguments and doesn't return any values. We will learn about arguments and return statements later in this tutorial.

Calling a Function in Python

In the above example, we have declared a function named `greet()`.

```
def greet():
    print('Hello World!')
```

Now, to use this function, we need to call it.

Here's how we can call the `greet()` function in Python.

```
# call the function
greet()
```

Example: Python Function

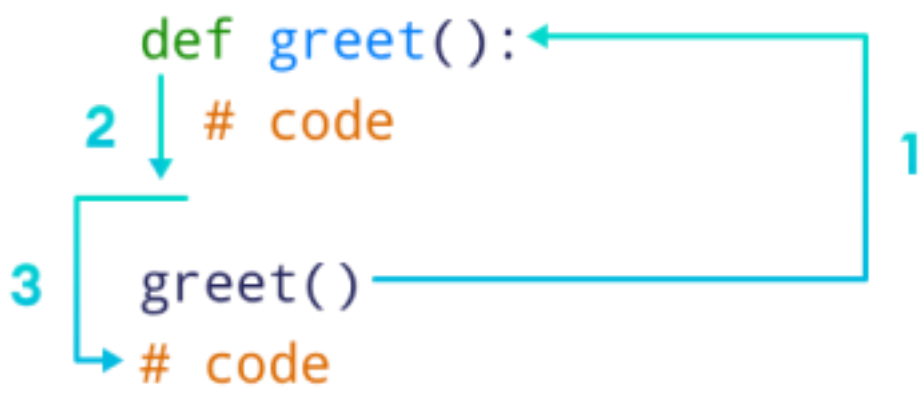
```
def greet():
    print('Hello World!')

# call the function
greet()

print('Outside function')
```

Output

```
Hello World!
Outside function
```

In the above example, we have created a function named `greet()`. Here's how

the program works:



Working of Python Function

Here,

- When the function is called, the control of the program goes to the function definition.
- All codes inside the function are executed.
- The control of the program jumps to the next statement after the function call.

Python Function Arguments

As mentioned earlier, a function can also have arguments. An argument is a value that is accepted by a function. For example,

```
# function with two arguments
def add_numbers(num1, num2):
```

```
    sum = num1 + num2
    print('Sum: ',sum)

# function with no argument
def add_numbers():
    # code
```

If we create a function with arguments, we need to pass the corresponding values while calling them. For example,

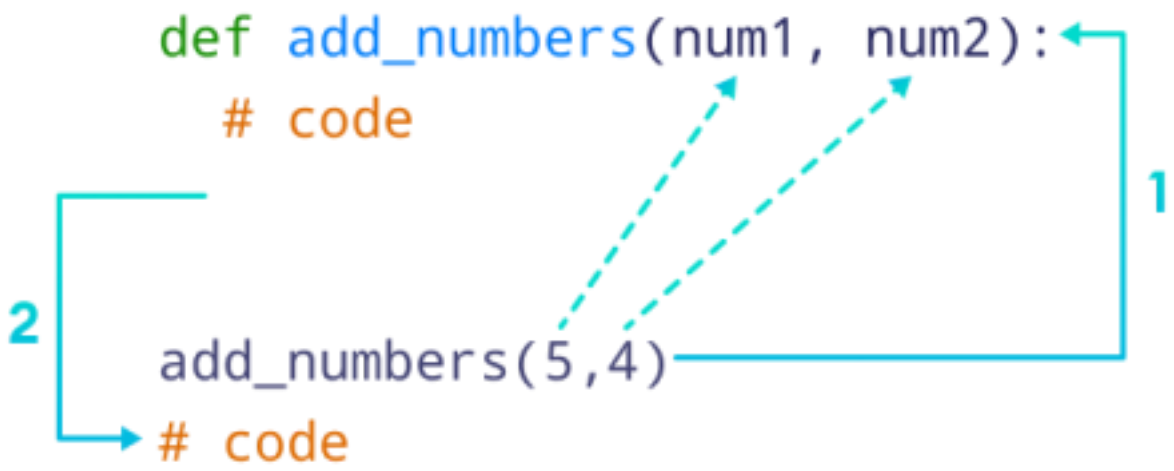
```
# function call with two values
add_numbers(5, 4)
# function call with no value
add_numbers()
```

Here, `add_numbers(5, 4)` specifies that arguments `num1` and `num2` will get values 5 and 4 respectively.

Example 1: Python Function Arguments

```
# function with two arguments
def add_numbers(num1, num2):
    sum = num1 + num2
    print("Sum: ",sum)

# function call with two values
add_numbers(5, 4)

# Output: Sum: 9
```

In the above example, we have created a function named `add_numbers()` with arguments: `num1` and `num2`.



Python Function with Arguments

We can also call the function by mentioning the argument name as:

```
add_numbers(num1 = 5, num2 = 4)
```

In Python, we call it Keyword Argument (or named argument). The code above is equivalent to

```
add_numbers(5, 4)
```

The return Statement in Python

A Python function may or may not return a value. If we want our function to return some value to a function call, we use the `return` statement. For

example,

```
def add_numbers():
    ...
    return sum
```

Here, we are returning the variable `sum` to the function call.
Note: The `return` statement also denotes that the function has ended. Any code after return is not executed.

Example 2: Function return Type

```
# function definition
def find_square(num):
    result = num * num
    return result

# function call
square = find_square(3)

print('Square:',square)

# Output: Square: 9
```

In the above example, we have created a function named `find_square()`. The function accepts a number and returns the square of the number.

```
def find_square(num):
    # code
    return result

Square = find_square(3)
# code
```

Working of functions in Python

Example 3: Add Two Numbers

```
# function that adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum

# calling function with two values
result = add_numbers(5, 4)

print('Sum: ', result)

# Output: Sum: 9
```

Python Library Functions

In Python, standard library functions are the built-in functions that can be used directly in our program. For example,

- `print()` - prints the string inside the quotation marks
- `sqrt()` - returns the square root of a number
- `pow()` - returns the power of a number

These library functions are defined inside the module. And, to use them we must include the module inside our program.

For example, `sqrt()` is defined inside the `math` module.

Example 4: Python Library Function

```
import math

# sqrt computes the square root
square_root = math.sqrt(4)

print("Square Root of 4 is",square_root)

# pow() comptes the power
power = pow(2, 3)

print("2 to the power 3 is",power)
```

Output

```
Square Root of 4 is 2.0
2 to the power 3 is 8
```

In the above example, we have used

- `math.sqrt(4)` - to compute the square root of 4
- `pow(2, 3)` - computes the power of a number i.e. 2^3

Here, notice the statement,

```
import math
```

Since `sqrt()` is defined inside the `math` module, we need to include it in our program.

Benefits of Using Functions

1. Code Reusable - We can use the same function multiple times in our program which makes our code reusable. For example,

```
# function definition
def get_square(num):
    return num * num

for i in [1,2,3]:
    # function call
    result = get_square(i)
    print('Square of',i, '=',result)
```

Output

```
Square of 1 = 1
Square of 2 = 4
Square of 3 = 9
```

In the above example, we have created the function named `get_square()` to calculate the square of a number. Here, the function is used to calculate the square of numbers from 1 to 3.

Hence, the same method is used again and again.

2. Code Readability - Functions help us break our code into chunks to make our program readable and easy to understand.

Python dictionary is an ordered collection (starting from Python 3.7) of items. It stores elements in key/value pairs. Here, keys are unique identifiers that are associated with each value.

Let's see an example,

If we want to store information about countries and their capitals, we can create a dictionary with country names as keys and capitals as values.

Keys Values
Nepal Kathmandu

Italy Rome

England London

Create a dictionary in Python

capital_city = {"Nepal": "Kathmandu", "Italy": "Rome", "England": "London"}

print(capital_city)

In the above example, we have created a dictionary named `capital_city`. Here,

1. Keys are `"Nepal"`, `"Italy"`, `"England"`
2. Values are `"Kathmandu"`, `"Rome"`, `"London"`

Note: Here, keys and values both are of string type. We can also have keys and values of different data types.

Example 1: Python Dictionary

```
# dictionary with keys and values of different data types
numbers = {1: "One", 2: "Two", 3: "Three"}
print(numbers)
```

Add Elements to a Python Dictionary

We can add elements to a dictionary using the name of the dictionary with `[]`. For example,

```
capital_city = {"Nepal": "Kathmandu", "England": "London"}
print("Initial Dictionary: ",capital_city)

capital_city["Japan"] = "Tokyo"

print("Updated Dictionary: ",capital_city)
```

In the above example, we have created a dictionary named `capital_city`. Notice the line,

```
capital_city["Japan"] = "Tokyo"
```

Here, we have added a new element to `capital_city` with key: `Japan` and value: `Tokyo`.

Change Value of Dictionary

We can also use `[]` to change the value associated with a particular key. For example,

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print("Initial Dictionary: ", student_id)

student_id[112] = "Stan"

print("Updated Dictionary: ", student_id)
```

Accessing Elements from Dictionary

In Python, we use the keys to access their corresponding values. For example,

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}

print(student_id[111]) # prints Eric
print(student_id[113]) # prints Butters
```

Here, we have used the keys to access their corresponding values.

If we try to access the value of a key that doesn't exist, we'll get an error. For example,

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print(student_id[211])

# Output: KeyError: 211
```

Removing elements from Dictionary

We use the `del` statement to remove an element from the dictionary. For example,

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
```

```
print("Initial Dictionary: ", student_id)

del student_id[111]

print("Updated Dictionary ", student_id)
```

Output

```
Initial Dictionary: {111: 'Eric', 112: 'Kyle', 113: 'Butters'}
Updated Dictionary {112: 'Kyle', 113: 'Butters'}
```

Here, we have created a dictionary named `student_id`. Notice the code,

```
del student_id[111]
```

The `del` statement removes the element associated with the key `111`. We can also delete the whole dictionary using the `del` statement,

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}

# delete student_id dictionary
del student_id

print(student_id)
# Output: NameError: name 'student_id' is not defined
```

We are getting an error message because we have deleted the `student_id` dictionary and `student_id` doesn't exist anymore.

Python Dictionary Methods

Methods that are available with a dictionary are tabulated below. Some of them have already been used in the above examples.

Funct ion	Description
all()	Return `True` if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return `True` if any key of the dictionary is true. If the dictionary is empty, return `False`.
len()	Return the length (the number of items) in the dictionary.
sorte d()	Return a new sorted list of keys in the dictionary.
value s()	Removes all items from the
clear( )	
keys( )	

dictionary. Returns a new Returns a new object of the

object of the dictionary's keys. dictionary's values

Dictionary Membership Test

We can test if a `key` is in a dictionary or not using the keyword `in`. Notice that the membership test is only for the `keys` and not for the `values`.

```
# Membership Test for Dictionary Keys
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True
print(1 in squares) # prints True

print(2 not in squares) # prints True

# membership tests for key only not value
print(49 in squares) # prints false
```

Output

```
True
True
False
```

Iterating Through a Dictionary

We can iterate through each key in a dictionary using a `for` loop.

```
# Iterating through a Dictionary
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
```

Output

```
1
9
25
49
81
```

Here, we have iterated through each key in the `squares` dictionary using the `for` loop.

In Python, **lists** are used to store multiple data at once. For example,

Suppose we need to record the ages of 5 students. Instead of creating 5 separate variables, we can simply create a list:

Create a Python List

A list is created in Python by placing items inside `[]`, separated by commas .

For example,

```
# A list with 3 integers

numbers = [1, 2, 5]


print(numbers)


# Output: [1, 2, 5]
```

Here, we have created a list named `numbers` with 3 integer items.

A list can have any number of items and they may be of different types (integer, float, string, etc.). For example,

```
# empty list

my_list = []


# list with mixed data types

my_list = [1, "Hello", 3.4]
```

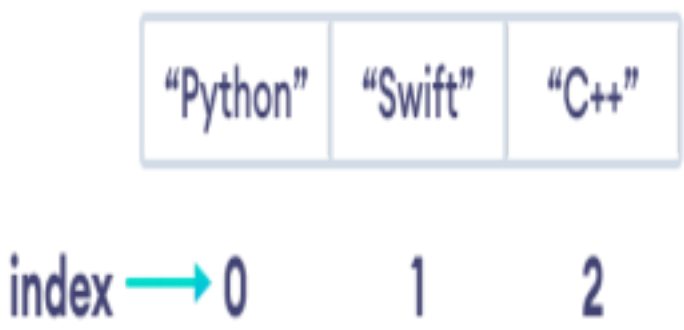
Access Python List Elements

In Python, each item in a list is associated with a number. The number is known as a list index.

We can access elements of an array using the index number (0, 1, 2 …). For example,

```
languages = ["Python", "Swift", "C++"]


# access item at index 0

print(languages[0]) # Python


# access item at index 2

print(languages[2]) # C++
```

In the above example, we have created a list named `languages`.



List Indexing in Python

Here, we can see each list item is associated with the index number. And, we have used the index number to access the items.

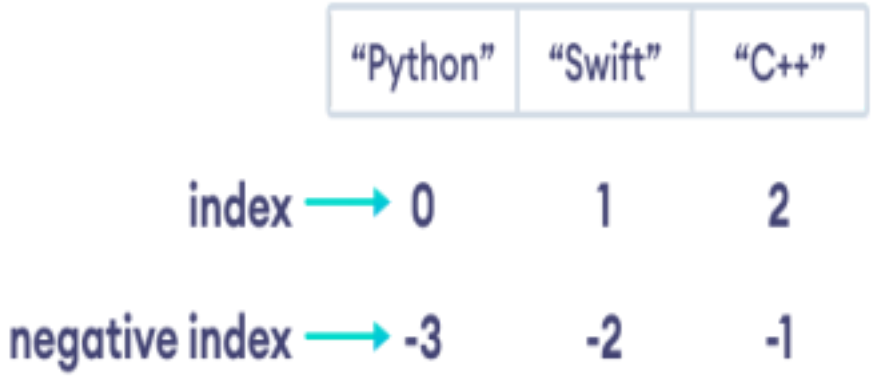
Note: The list index always starts with 0. Hence, the first element of a list is present at index 0, not 1.

Negative Indexing in Python

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

Let's see an example,

```
languages = ["Python", "Swift", "C++"]



# access item at index 0

print(languages[-1]) # C++
# access item at index 2

print(languages[-3]) # Python
```



Python Negative Indexing

Note: If the specified index does not exist in the list, Python throws the `IndexError` exception.

Slicing of a Python List

In Python it is possible to access a section of items from the list using the slicing operator `:`, not just a single item. For example,

```
# List slicing in Python
my_list = ['p','r','o','g','r','a','m','i','z']
```

```
# items from index 2 to index 4
```

```
print(my_list[2:5])
```

```
# items from index 5 to end
```

```
print(my_list[5:])
```

```
# items beginning to end
print(my_list[:])
```

Output

```
['o', 'g', 'r']
```

```
['a', 'm', 'i', 'z']
```

```
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

Here,

- `my_list[2:5]` returns a list with items from index 2 to index 4.
- `my_list[5:]` returns a list with items from index 1 to the end.
- `my_list[:]` returns all list items

Note: When we slice lists, the start index is inclusive but the end index is exclusive.

Add Elements to a Python List

Python List provides different methods to add items to a

list. 1. Using append()

The append() method adds an item at the end of the list. For example,

```
numbers = [21, 34, 54, 12]


print("Before Append:", numbers)


# using append method

numbers.append(32)


print("After Append:", numbers)
```

Output

```
Before Append: [21, 34, 54, 12]
After Append: [21, 34, 54, 12, 32]
```

In the above example, we have created a list named `numbers`. Notice the line,

```
numbers.append(32)
```

Here, `append()` adds 32 at the end of the array.

2. Using extend()

We use the extend() method to add all items of one list to another. For

example,

```
prime_numbers = [2, 3, 5]

print("List1:", prime_numbers)



even_numbers = [4, 6, 8]

print("List2:", even_numbers)



# join two lists

prime_numbers.extend(even_numbers)



print("List after append:", prime_numbers)
```

Output

```
List1: [2, 3, 5]

List2: [4, 6, 8]

List after append: [2, 3, 5, 4, 6, 8]
```

In the above example, we have two lists named `prime_numbers` and `even_numbers`. Notice the statement,

```
prime_numbers.extend(even_numbers)
```

Here, we are adding all elements of `even_numbers` to `prime_numbers`.

Change List Items

Python lists are mutable. Meaning lists are changeable. And, we can change items of a list by assigning new values using `=` operator. For example,

```
languages = ['Python', 'Swift', 'C++']


# changing the third item to 'C'

languages[2] = 'C'


print(languages) # ['Python', 'Swift', 'C']
```

Here, initially the value at index 3 is `'C++'`. We then changed the value to `'C'` using

```
languages[2] = 'C'
```

Remove an Item From a List

1. Using del()

In Python we can use the del statement to remove one or more items from a list. For example,

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']


# deleting the second item

del languages[1]

print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust', 'R']


# deleting the last item

del languages[-1]

print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust']


# delete first two items

del languages[0 : 2] # ['C', 'Java', 'Rust']

print(languages)
```

2. Using remove()

We can also use the remove() method to delete a list item. For example,

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']


# remove 'Python' from the list

languages.remove('Python')
```

```
print(languages) # ['Swift', 'C++', 'C', 'Java', 'Rust', 'R']
```

Here, `languages.remove('Python')` removes `'Python'` from the `languages` list.

Python List Methods

Python has many useful list methods that makes it really easy to work with lists.

Method Description

append() add an item to the end of the list
extend() add items of lists and other iterables to the end of the list insert()

inserts an item at the specified index

remove() removes item present at the given index pop() returns and

removes item present at the given index clear() removes all items

from the list

index() returns the index of the first matched item count() returns

the count of the specified item in the list sort() sort the list in

ascending/descending order reverse() reverses the item of the

list

copy() returns the shallow copy of the list

Iterating through a List

We can use the for loop to iterate over the elements of a list. For example,

```
languages = ['Python', 'Swift', 'C++']

# iterating through the list
for language in languages:
    print(language)
```

Output

```
Python

Swift

C++
```

Check if an Item Exists in the Python List

We use the `in` keyword to check if an item exists in the list or not. For example,

```
languages = ['Python', 'Swift', 'C++']


print('C' in languages) # False
print('Python' in languages) # True
```

Here,

- `'C'` is not present in `languages`, `'C' in languages` evaluates to `False`.
- `'Python'` is present in `languages`, `'Python' in languages` evaluates to `True`.

Python List Length

In Python, we use the `len()` function to find the number of elements present in a list. For example,

```
languages = ['Python', 'Swift', 'C++']


print("List: ", languages)


print("Total Elements: ", len(languages)) # 3
```

Output

```
List: ['Python', 'Swift', 'C++']
Total Elements: 3
```

Python List Comprehension

List comprehension is a concise and elegant way to create lists.

A list comprehension consists of an expression followed by the for statement inside square brackets.

Here is an example to make a list with each item being increasing by power of 2.

```
numbers = [number*number for number in range(1, 6)]


print(numbers)


# Output: [1, 4, 9, 16, 25]
```

In the above example, we have used the list comprehension to make a list with each item being increased by power of 2. Notice the code,

```
[number*x for x in range(1, 6)]
```

The code above means to create a list of `number*number` where `number` takes values from 1 to 5

The code above,

```
numbers = [x*x for x in range(1, 6)]
```

is equivalent to

```
numbers = []


for x in range(1, 6):

    numbers.append(x * x)
```

A **tuple** in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses `()`, separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

```
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)
# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

Run Code

Output

```
()
(1, 2, 3)
(1, 'Hello', 3.4)
('mouse', [8, 4, 6], (1, 2, 3))
```

In the above example, we have created different types of tuples and stored different data items inside them.

As mentioned earlier, we can also create tuples without using parentheses:

```
my_tuple = 1, 2, 3
my_tuple = 1, "Hello", 3.4
```

Create a Python Tuple With one Element

In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.
We will need a trailing comma to indicate that it is a tuple,

```
var1 = ("Hello") # string
var2 = ("Hello",) # tuple
```

We can use the `type()` function to know which class a variable or a value belongs to.

```
var1 = ("hello")
```

```
print(type(var1)) # <class 'str'>

# Creating a tuple having one element
var2 = ("hello",)
print(type(var2)) # <class 'tuple'>

# Parentheses is optional
var3 = "hello",
print(type(var3)) # <class 'tuple'>
```

Run Code

Here,

- `("hello")` is a string so `type()` returns `str` as class of `var1` i.e. `<class 'str'>`
- `("hello",)` and `"hello",` both are tuples so `type()` returns `tuple` as class of `var1` i.e. `<class 'tuple'>`

Access Python Tuple Elements

Like a list, each element of a tuple is represented by index numbers (0, 1, ...) where the first element is at index 0.

We use the index number to access tuple elements. For example,

1. Indexing

We can use the index operator `[]` to access an item in a tuple, where the index starts from 0.

So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range(6,7,... in this example) will raise an `IndexError`.

The index must be an integer, so we cannot use float or other types. This will result in `TypeError`.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.

```
# accessing tuple elements using indexing
letters = ("p", "r", "o", "g", "r", "a", "m", "i", "z")

print(letters[0]) # prints "p"
print(letters[5]) # prints "a"
```

Run Code

In the above example,

- `letters[0]` - accesses the first element
- `letters[5]` - accesses the sixth element

2. Negative Indexing

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on.

For example,

```
# accessing tuple elements using negative indexing
letters = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

print(letters[-1]) # prints 'z'
print(letters[-3]) # prints 'm'
```

Run Code

In the above example,

- `letters[-1]` - accesses last element
- `letters[-3]` - accesses third last element

3. Slicing

We can access a range of items in a tuple by using the slicing operator colon `:`.

```
# accessing tuple elements using slicing
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# elements 2nd to 4th index
print(my_tuple[1:4]) # prints ('r', 'o', 'g')

# elements beginning to 2nd
print(my_tuple[:-7]) # prints ('p', 'r')
```

```
# elements 8th to end
print(my_tuple[7:]) # prints ('i', 'z')

# elements beginning to end
print(my_tuple[:]) # Prints ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

Run Code

Output

```
('r', 'o', 'g')
('p', 'r')
('i', 'z')
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

Here,

- `my_tuple[1:4]` returns a tuple with elements from index 1 to index 3. • `my_tuple[:-7]` returns a tuple with elements from beginning to index 2.
- `my_tuple[7:]` returns a tuple with elements from index 7 to the end.
- `my_tuple[:]` returns all tuple items.

Note: When we slice lists, the start index is inclusive but the end index is exclusive.

Python Tuple Methods

In Python ,methods that add items or remove items are not available with tuple. Only the following two methods are available.

Some examples of Python tuple methods:

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p')) # prints 2
print(my_tuple.index('l')) # prints 3
```

Run Code

Here,

- `my_tuple.count('p')` - counts total number of `'p'` in `my_tuple`
- `my_tuple.index('l')` - returns the first occurrence of `'l'` in `my_tuple`

Iterating through a Tuple in Python

We can use the for loop to iterate over the elements of a tuple. For example,

```
languages = ('Python', 'Swift', 'C++')

# iterating through the tuple
for language in languages:
    print(language)
```

Run Code

Output

```
Python
Swift
C++
```

Check if an Item Exists in the Python Tuple

We use the `in` keyword to check if an item exists in the tuple or not. For example,

```
languages = ('Python', 'Swift', 'C++')

print('C' in languages) # False
print('Python' in languages) # True
```

Run Code

Here,

- `'C'` is not present in `languages`, `'C' in languages` evaluates to `False`.
- `'Python'` is present in `languages`, `'Python' in languages` evaluates to `True`.

Advantages of Tuple over List in Python

Since tuples are quite similar to lists, both of them are used in similar situations.

However, there are certain advantages of implementing a tuple over a list:

- We generally use tuples for heterogeneous (different) data types and lists for homogeneous (similar) data types.
- Since tuples are immutable, iterating through a tuple is faster than with a list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

A **set** is a collection of unique data. That is, elements of a set cannot be duplicate. For example,

Suppose we want to store information about student IDs. Since student IDs cannot be duplicate, we can use a set.

Create a Set in Python

In Python, we create sets by placing all the elements inside curly braces `{}`, separated by comma.

A set can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

Let's see an example,

```
# create a set of integer type

student_id = {112, 114, 116, 118, 115}

print('Student ID:', student_id)


# create a set of string type

vowel_letters = {'a', 'e', 'i', 'o', 'u'}

print('Vowel Letters:', vowel_letters)


# create a set of mixed data types

mixed_set = {'Hello', 101, -2, 'Bye'}

print('Set of mixed data types:', mixed_set)
```

Run Code

Output

```
Student ID: {112, 114, 115, 116, 118}

Vowel Letters: {'u', 'a', 'e', 'i', 'o'}

Set of mixed data types: {'Hello', 'Bye', 101, -2}
```

In the above example, we have created different types of sets by placing all the elements inside the curly braces `{}`.

Note: When you run this code, you might get output in a different order. This is

because the set has no particular order.

Create an Empty Set in Python

Creating an empty set is a bit tricky. Empty curly braces `{}` will make an empty dictionary in Python.

To make a set without any elements, we use the set() function without any argument. For example,

```
# create an empty set

empty_set = set()



# create an empty dictionary

empty_dictionary = { }
# check data type of empty_set

print('Data type of empty_set:', type(empty_set))



# check data type of dictionary_set

print('Data type of empty_dictionary', type(empty_dictionary))
```

Run Code

Output

```
Data type of empty set: <class 'set'>
```

```
Data type of empty dictionary <class 'dict'>
```

Here,

- `empty_set` - an empty set created using `set()`
- `empty_dictionary` - an empty dictionary created using `{}`

Finally we have used the `type()` function to know which class `empty_set` and `empty_dictionary` belong to.

Duplicate Items in a Set

Let's see what will happen if we try to include duplicate items in a set.

```
numbers = {2, 4, 6, 6, 2, 8}

print(numbers) # {8, 2, 4, 6}
```

Run Code

Here, we can see there are no duplicate items in the set as a set cannot contain duplicates.

Add and Update Set Items in Python

Sets are mutable. However, since they are unordered, indexing has no meaning.

We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

Add Items to a Set in Python

In Python, we use the `add()` method to add an item to a set. For example, `numbers = {21, 34, 54, 12}`

```
print('Initial Set:',numbers)
```

```
# using add() method

numbers.add(32)
print('Updated Set:', numbers)
```

Run Code

Output

```
Initial Set: {34, 12, 21, 54}

Updated Set: {32, 34, 12, 21, 54}
```

In the above example, we have created a set named `numbers`. Notice the line,

```
numbers.add(32)
```

Here, `add()` adds 32 to our set.

Update Python Set

The `update()` method is used to update the set with items other collection types (lists, tuples, sets, etc). For example,

```
companies = {'Lacoste', 'Ralph Lauren'}

tech_companies = ['apple', 'google', 'apple']


companies.update(tech_companies)


print(companies)


# Output: {'google', 'apple', 'Lacoste', 'Ralph Lauren'}
```

Run Code

Here, all the unique elements of `tech_companies` are added to the `companies` set.

Remove an Element from a Set

We use the `discard()` method to remove the specified element from a set. For example,

```
languages = {'Swift', 'Java', 'Python'}


print('Initial Set:',languages)


# remove 'Java' from a set

removedValue = languages.discard('Java')


print('Set after remove():', languages)
```

Run Code

Output

```
Initial Set: {'Python', 'Swift', 'Java'}

Set after remove(): {'Python', 'Swift'}
```

Here, we have used the `discard()` method to remove `'Java'` from the `languages` set.

Built-in Functions with Set

Built-in functions like `all()`, `any()`, `enumerate()`, `len()`, `max()`, `min()`, `sorted()`,

`sum()` etc. are commonly used with sets to perform different tasks.

Function Description

all() Returns `True` if all elements of the set are true (or if the set is empty).

any() Returns `True` if any element of the set is true. If the set is empty, returns `False`.

enumer ate() the index and value for all the items of the set as a pair.

Returns an enumerate object. It contains

len() Returns the length (the number of items) in the set.

max() Returns the largest item in the set.

min() Returns the smallest item in the set.

sorted() Returns a new sorted list from elements in the set(does not sort the set itself).

sum() Returns the sum of all elements in the set.

Iterate Over a Set in Python

```
fruits = {"Apple", "Peach", "Mango"}


# for loop to access each fruits

for fruit in fruits:

    print(fruit)
```

Run Code

Output

```
Mango

Peach
Apple
```

Find Number of Set Elements

We can use the `len()` method to find the number of elements present in a Set.

For example,

```
even_numbers = {2,4,6,8}

print('Set:',even_numbers)
```

```
# find number of elements

print('Total Elements:', len(even_numbers))
```

Run Code

Output

```
Set: {8, 2, 4, 6}

Total Elements: 4
```

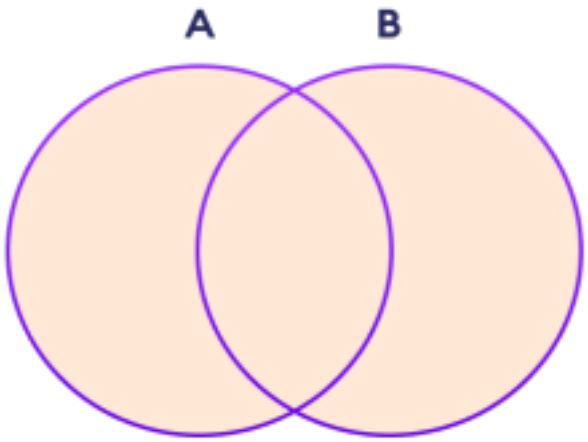
Here, we have used the `len()` method to find the number of elements present in a Set.

Python Set Operations

Python Set provides different built-in methods to perform mathematical set operations like union, intersection, subtraction, and symmetric difference.

Union of Two Sets

The union of two sets A and B include all the elements of set A and B.



Set Union in Python

We use the `|` operator or the `union()` method to perform the set union operation. For example,

```
# first set

A = {1, 3, 5}



# second set

B = {0, 2, 4}



# perform union operation using |
print('Union using |:', A | B)



# perform union operation using union()

print('Union using union():', A.union(B))
```

Run Code

Output

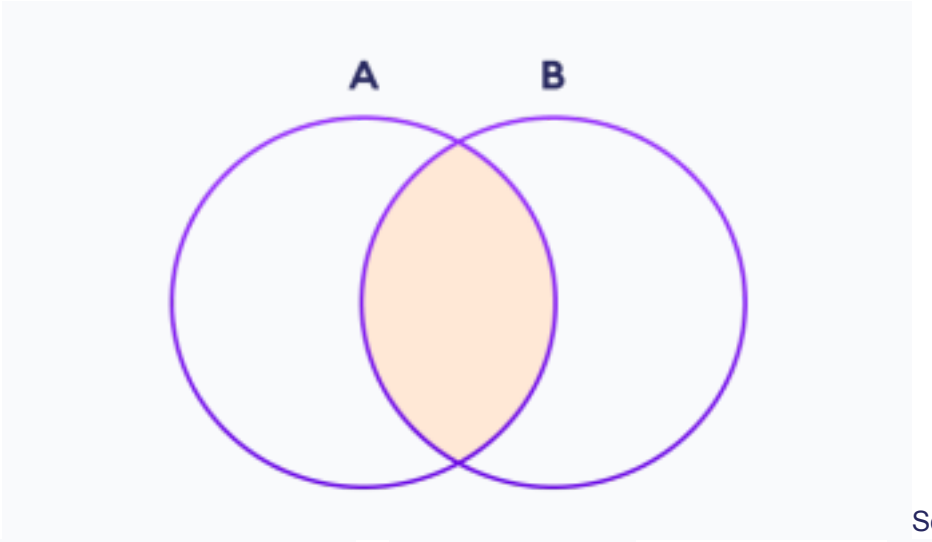
```
Union using |: {0, 1, 2, 3, 4, 5}
```

```
Union using union(): {0, 1, 2, 3, 4, 5}
```

Note: `A|B` and `union()` is equivalent to `A ∪ B` set operation.

Set Intersection

The intersection of two sets A and B include the common elements between set A and B.



Set Intersection in Python

In Python, we use the `&` operator or the `intersection()` method to perform the set intersection operation. For example,

```
# first set
```

```
A = {1, 3, 5}

# second set
B = {1, 2, 3}

# perform intersection operation using &
print('Intersection using &:', A & B)

# perform intersection operation using intersection()
print('Intersection using intersection():', A.intersection(B))
```

Run Code

Output

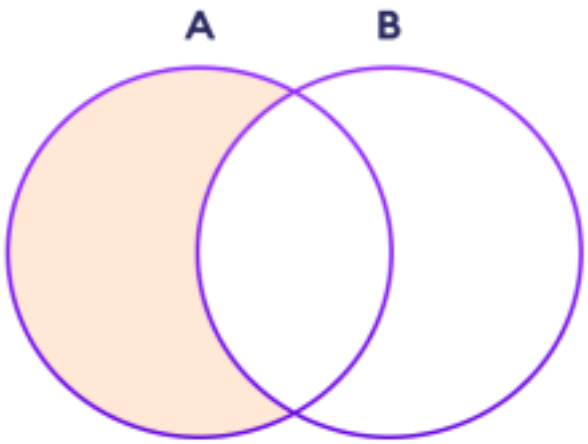
```
Intersection using &: {1, 3}
Intersection using intersection(): {1, 3}
```

Note: `A&B` and `intersection()` is equivalent to `A ∩ B` set operation.

Difference between Two Sets

The difference between two sets A and B include elements of set A that are

not present on set B.



Set Difference in Python

We use the `-` operator or the `difference()` method to perform the difference between two sets. For example,

```
# first set
A = {2, 3, 5}


# second set

B = {1, 2, 6}


# perform difference operation using &

print('Difference using &:', A - B)


# perform difference operation using difference()
```

```
print('Difference using difference():', A.difference(B))
```

Run Code

Output

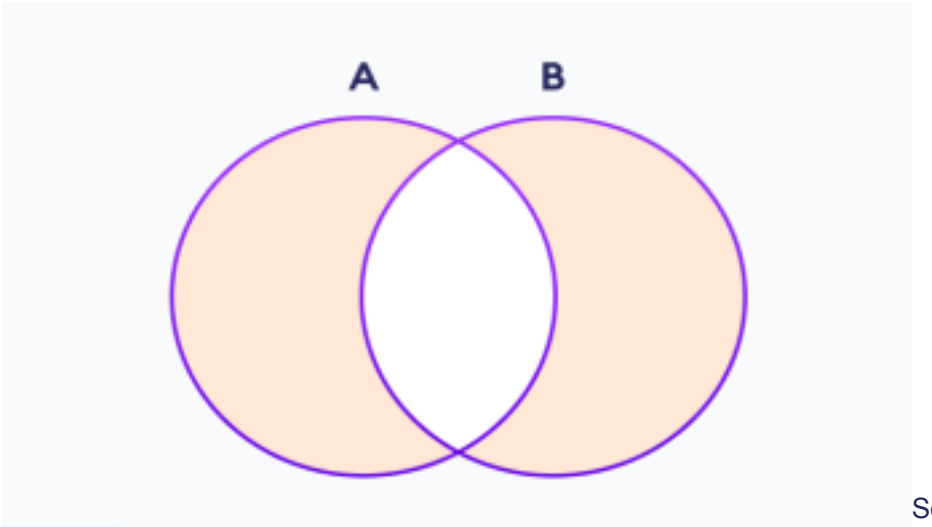
```
Difference using &: {3, 5}
```

```
Difference using difference(): {3, 5}
```

Note: `A - B` and `A.difference(B)` is equivalent to `A - B` set operation.

Set Symmetric Difference

The symmetric difference between two sets A and B includes all elements of A and B without the common elements.



Set Symmetric Difference in Python

In Python, we use the `^` operator or the `symmetric_difference()` method to

perform symmetric difference between two sets. For example,

```
# first set

A = {2, 3, 5}


# second set

B = {1, 2, 6}


# perform difference operation using &

print('using ^:', A ^ B)


# using symmetric_difference()

print('using symmetric_difference():', A.symmetric_difference(B))
```

Run Code

Output

```
using ^: {1, 3, 5, 6}

using symmetric_difference(): {1, 3, 5, 6}
```

Check if two sets are equal

We can use the `==` operator to check whether two sets are equal or not. For example,

```
# first set

A = {1, 3, 5}


# second set

B = {3, 5, 1}


# perform difference operation using &

if A == B:

    print('Set A and Set B are equal')

else:

    print('Set A and Set B are not equal')
```

Run Code

Output

```
Set A and Set B are equal
```

In the above example, `A` and `B` have the same elements, so the condition

```
if A == B
```

evaluates to `True`. Hence, the statement `print('Set A and Set B are equal')` inside the `if` is executed.

Other Python Set Methods

There are many set methods, some of which we have already used above.

Here is a list of all the methods that are available with the set objects:

Method Description

add() Adds an element to the set

clear() Removes all elements from the set

copy() Returns a copy of the set

difference() Returns the difference of two or more sets as a new set

difference_update() Removes all elements of another set from this set

discard() Removes an element from the set if it is a member. (Do nothing if the element is not in set)

intersection() Returns the intersection of two sets as a new set

intersection_update() Updates the set with the intersection of itself and another

isdisjoint() Returns `True` if two sets have a null intersection issubset() Returns

`True` if another set contains this set issuperset() Returns `True` if this set

contains another set

pop() Removes and returns an arbitrary set element. Raises `KeyError` if the set is empty

remove() Removes an element from the set. If the element is not a member, raises a `KeyError`

symmetric_difference() Returns the symmetric difference of two sets as a new set

symmetric_difference_u pdate() Updates a set with the symmetric difference of itself and another

union() Returns the union of sets in a new set update() Updates the set with the

union of itself and others

Tasks

Task1: Write a function, which takes input as :

1. Triangle
2. Rectangle

Choose the number.

The task is to implement the printing of star pattern.

Example Output:

```
      *
    * *
  * *
```

Or

```
*******
*
*
*******
```

Task 2: A) Create a dictionary and apply the following methods:

1) Print the dictionary items

2) access items

3) use get()

4)change values
5) use len()

B) Create a tuple and perform the following methods:

1) Add items

2) len()

3) check for item in tuple

4)Access items

Task3: Write a program to calculate Volume of rod up to 3 floating points. You should take float input from the user. Take r and h from users as float values.

Formula: $V = \pi \times r^2 \times h$

Note: use proper value of pi

Task4: Write a program to create a menu with the following options

1. TO PERFORM ADDITION

2. TO PERFORM SUBTRACTION

3. TO PERFORM MULTIPLICATION

4. TO PERFORM DIVISION Accepts users input and perform the operation accordingly. Use functions with arguments.

Task5: Write a python program to check whether the given string is palindrome or not.

Task6 : Write a python program to find factorial of a given number using functions

Task 7: Write a Python function that takes two lists, prints them side by side and returns True if they are equal otherwise false

Task8: WriteaPythonprogramtocheck ifagivenset isasupersetof itselfanda supersetofanothergivenset.