Hello, World!

python

```
print("Hello, World!")
```

Variables and Data Types:

Variables are used to store data.
Common data types include:

- int (integers)
- float (floating-point numbers)
- str (strings)
- bool (Boolean values)

```
age = 25
name = "Akshay"
is_student = True
```

Input and Output:

Use input() to get user input and print() to display output.

```
name = input("Enter your name: ")
print("Hello, " + name)
```

Conditional Statements (if, elif, else):

The if statement is used to execute a block of code only if a condition is true.

```
if condition:
    # Code to execute if the condition is true
```

Checking for a Condition:

In this example, we check if a number is greater than 10 and print a message accordingly.

```
number = 15
if number > 10:
    print("The number is greater than 10.")
```

Indentation:

- Python uses indentation (whitespace) to define blocks of code.
- Indentation is critical for proper code execution.
- Always use consistent indentation, such as four spaces or a tab.

The else statement is used to execute a block of code if the if condition is not true.
It is often used in conjunction with if.

```
number = 5
if number > 10:
    print("The number is greater than 10.")
else:
    print("The number is not greater than 10.")
```

Sometimes, you want to check multiple conditions in a specific order.
You can use the elif statement (short for "else if") for this purpose.

```
grade = 75
if grade >= 90:
    print("A")
elif grade >= 80:
    print("B")
elif grade >= 70:
    print("C")
else:
    print("D")
```

Nested if Statements:

You can nest if statements inside other if statements to create complex conditions.

```python
x = 10
if x > 5:
    if x < 15:
        print("x is between 5 and 15.")
```

```python
age = 18
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

Logical Operators:

Python supports logical operators like **and, or, and not** to create more complex conditions.

```python
age = 25
if age >= 18 and age <= 65:
    print("You are of working age.")
```

Indentation and Consistency:

- Pay close attention to indentation to ensure your code is properly structured.
- Consistent formatting enhances code readability.

Loops (for, while):

- for loops are used to iterate over a sequence.
- while loops execute as long as a condition is true.

```python
for i in range(5):
    print(i)
```

```python
x = 0
while x < 5:
    print(x)
    x += 1
```

while loop is used to repeatedly execute a block of code as long as a specified condition is True. Here's the basic syntax of a while loop:

```
while condition:
    # Code to be executed while the condition is True
    # This code block can contain one or more statements
```

The loop will continue to execute the code block as long as the condition remains True. Once the condition becomes False, the loop will exit, and the program will continue with the next statement after the while loop.

Example: Use while loop to print numbers from 1 to 5**

```
python
#counter=0
#print(counter)
#counter+=1

#print(counter)
#counter+=1

#print(counter)
#counter+=1
.
.
.
#n times
```

```
num=1
while(num<=5):
  print(num)
  num+=1
```

Question: Take an input from user and print all the alternate numbers from 1 till input

```
number = int(input("Please input a number - "))
i = 1
while i <= number:
    print(i)
    i += 2
```

Question: Take an input from the user for start and end values and print every third number

```
start = int(input())
end = int(input())
while start <= end:
    print(start)
    start += 3
```

Question: Take an input from the user for start and end values. Print all even numbers between them.

- Start at the number and print if the number is divisible by 2, increment number by 1

```
start = int(input())
end = int(input())
loop_count = 0
while start <= end:
    loop_count += 1
    if start % 2 == 0:
        print(start)
    start += 1
print("Loop was run", loop_count,"times!")
```

- If number is even, start from number, if odd, start from number+1, print number, increment by 2

```
start = int(input())
end = int(input())
if start % 2 == 0:
    i = start
else:
    i = start +1
loop_count = 0
while i <= end:
    loop_count += 1
    print(i)
    i += 2
print("Loop was run", loop_count,"times!")
```

- Second answer is more optimized, since the loop runs lesser number of times.

Question: You are a bowler and you have to bowl 1 over. (6 balls) "Bowling ball number - ball_number

```
ball_number = 1
while ball_number <= 6:
    print("Bowling ball number -", ball_number)
    ball_number += 1
```

Question: Print all numbers from 1 to N (input from user) in a single line.

- Use `end =" "` to print space separated values.

```
number = int(input())
i = 1
while i <= number:
    print(i, end=" ")
    i += 1
```

Question: Given a number, print it's multiplication table!

```
number = int(input())
i = 1
while i <= 10:
    print(number*i, end=" ")
    i += 1
```

Question:
1. Take the number of test_cases (count) from a user. (Count of numbers)
2. Take int(count) inputs from the user
3. Print multiplication table for all these inputs!

```python
# 3
# 4
# multiplication table of 4
# 5
# multiplication table of 5
# 6
# multiplication table of 6
```

```python
test_cases = int(input())
t = 1
while t <= test_cases:
    number = int(input("Please enter the number -"))
    i = 1
    print("Printing multiplication table of -", number)
    while i <= 10:
        print(number*i, end=" ")
        i += 1
    print("\n")
    t += 1
```

Python range() function :

```
x = range(5)
for n in x:
  print(n)
```

Output : 0 1 2 3 4

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

```
range(start, stop, step)
```

- start: **Optional**. An integer number specifying at which position to start. Default is 0
- stop: **Required**. An integer number specifying at which position to stop (not included).
- step: **Optional**. An integer number specifying the incrementation. Default is 1

```
x = range(3, 6)
for n in x:
  print(n)
```

Output : 3 4 5

Example : Create a sequence of numbers from 4 to 20, but increment by 2 instead of 1:

```
x = range(4, 21, 2)
for n in x:
  print(n)
```

Example: Print all numbers from 10 to 1

```
for i in range(10, 0, -1):
    print(i)
```

Q: Write a program that continuously asks the user to provide an input number.The program should stop only when the user provides 5.Once the user provides 5. The program should print the number of times an input was provided.Assume that user always provides a number!

```
counter = 0
while True:
    number = int(input())
    counter += 1

    if number == 5:
        break
print("You provided", counter, "inputs!")
```

Example : Write a program to print a N x N matrix of *

```
n = int(input())
for i in range(n):
    for j in range(n):
        print("*", end = " ")
    print()
```

```
3

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

Example : Write a program to print a multiplication table till an input N.

```
number = int(input())
for i in range(1, number + 1):
    for j in range(1, 11):
        print(i*j, end = " ")
    print()
```

Example Input:
3

Example Output:

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
```

Functions in python:

```
name = input()
def greet(name):
    print("Hello", name)
```

Ternary Operator :

```
if 4>5:
   a = 50
else:
   a = 40
print(a)
```

40

Instead we can also compress this entire thing into a single line using ternary operator. It simply allows testing a condition in a single line replacing the multiline if-else making the code compact.

```
'''
Syntax:
[on_true] if [expression] else [on_false]
'''
a = 50 if 4 > 5 else 40
print(a)
```

Lists :

- Lists are an ordered collection of data.
- It is a data structure that can store multiple values.
- Lists have no limit on how many values it can store.

```
# creating a list -> []
# They store comma separated values.
runs_virat = [67, 54, 12, 34, 77, 89, 101]
```

```
Output : runs_virat
[67, 54, 12, 34, 77, 89, 101]
```

```
length_runs_virat = len(runs_virat)
```

How do I calculate runs scored by virat in the last match?

```
runs_virat[len(runs_virat) - 1]
```

```
# Python makes it simpler by using negative indexing.
runs_virat[-1]
```

```
# Small question - What does runs_virat[-len(runs_virat)] give?
runs_virat[-len(runs_virat)] # runs_virat[0]
```

Can we add more elements to the list?

# I want to add more runs to this
runs = [45, 67, 89, 12, 34, 56, 100]

```
# append is a method on list datatype
runs.append(71)
```

Can I add an element at a particular index?

```
runs.insert(0, 25)
```

```
runs
[25, 45, 67, 89, 12, 34, 56, 100, 71]
```

Let's say we have a variable new_runs = [130, 69, 92] and I want to add this to the end of the runs list. How can I do this?

```
new_runs = [130, 69, 92]
runs.extend(new_runs)
```

```
# We can also use "+" instead of extend
runs = runs + [200, 250, 300]
```

Iterating over Lists :

```
# i -> iterator -> variable used to iterate
# range(5) -> iterable -> the collection of values on which we iterate
# print(i) -> iteration block -> bod of the for loop
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Lists are also iterable.

```
a = [56, 78, 89, 32, 101, 4]
for i in a:
    print(i)
```

```
length_runs = 0
average_runs = 0
sum_runs = 0
max_runs = 0
min_runs = 0
```

```
for i in runs_virat:
    sum_runs = sum_runs + i
    length_runs = length_runs + 1
```

```
average_runs = sum_runs / length_runs
```

```
for i in a:
    if i > max_runs:
        max_runs = i
```

```
max_runs
```

```
min_runs = max_runs
for i in a:
    if i < min_runs:
        min_runs = i
min_runs
```

We can also use in-built python functions.

```
sum_runs = sum(runs_virat)

average_runs = sum_runs / len(runs_virat)
average_runs


max(runs_virat)

min(runs_virat)
```

Calculate the sum of runs made by Virat in all matches with even index.

```
sum_even_runs = 0
for i in range(len(runs_virat)):
    if i % 2 == 0:
        sum_even_runs = sum_even_runs + runs_virat[i]
sum_even_runs
```

LIST SLICING :

Question : Given a list of all runs by Virat, create a new list of runs made in the last 5 matches. runs = [62, 85, 74, 10, 12, 101, 122, 99, 81, 55].

```
runs = [62, 85, 74, 10, 12, 101, 122, 99, 81, 55]
result = []
for i in range(len(runs) - 5, len(runs)):
    result.append(runs[i])
result
```

[101, 122, 99, 81, 55]

```
# Will have to reverse the list
runs = [62, 85, 74, 10, 12, 101, 122, 99, 81, 55]
result = []
for i in range(1, 6):
    result.append(runs[-i])
result
```

[55, 81, 99, 122, 101]

Question : Given a list of all runs by Virat, create a new list of runs made from the 3rd match to the 7th match. runs = [62, 85, 74, 10, 12, 101, 122, 99, 81, 55]

```
runs = [62, 85, 74, 10, 12, 101, 122, 99, 81, 55]
for i in range(2, 7):
    print(runs[i])
```

Question : Given a list of all runs by Virat, create a new list of runs made in the first 3 matches. runs = [62, 85, 74, 10, 12, 101, 122, 99, 81, 55]

```
runs = [62, 85, 74, 10, 12, 101, 122, 99, 81, 55]
for i in range(3):
    print(runs[i])
```

List Slicing:

The motivation behind the 5 questions above was to show learners the importance and need of list slicing.

Accessing a range of elements:

- We can access a range of elements using list slicing
- The syntax is as follows.
- The original list is not updated when slicing.

runs[x,y] -> This represents the list items from index **x (Included) to y (Excluded).**

Accessing odd indexed elements:

runs[0, len(runs), 2] -> Here 2 is the step size, i.e increase the index by 2 (by default the step size is 1).

- If we just provide the start value and not the end value, something like runs[2:] it will print all the elements from index 2 till the end of the list.
- If we just provide the end value and not the start value, something like runs[:5] it will print all the elements til index 5 excluding the index 5 (something similar to ranges).
- If we do not provide the start and the end value both, something like runs[::2], but we do provide a value in the step size, it will print all the elements at even indexes in list runs.

```
# We can use a step size of -1, to back traverse a list
runs[5:0:-1]
```

```
# Can go from the second last element to the 5th last
# with a jump of -1
runs[-2:-5:-1]
```

STRINGS in Python:

What is a String?

String in python are a sequence of characters.

What type of characters can we have?

- Small alphabets - [a - z]
- Capital Alphabets - [A - Z]
- Numbers - [0 - 9]
- White spaces
- Special characters

Question: Take a string input from the user. Reverse that string.

```
s = "any string"
s[::-1]
```

Question: Take a string input from the user. Return "PALINDROME" if the string is a palindrome, otherwise return "NOT A PALINDROME".

```
def check_palindrome(a):
    return "PALINDROME" if a == a[::-1] else "NOT A PALINDROME"
```

```
def check_palindrome(a):
    a = a.lower()
    return "PALINDROME" if a == a[::-1] else "NOT A PALINDROME"
```

Question: Take a string input from the user.Print the number of uppercase characters in that string.

```
def count_upper_case(a):
    count = 0
    for i in a:
        if i.isupper():
            count = count + 1
    return count
```

Question: Given a string of comma separated values. Convert it into a string of individual values.

Input:
"56,78,99,101"

Output:
[56, 78, 99, 101]

```
def easy_way(a):
    list_a = a.split(",")
    result = []
    for i in list_a:
        result.append(int(i))
    return result
```

- "".join() --> Input would be a list of strings --> Output would be a string.
- "Random_String".split("_") --> Argument is the splitting character --> Result would be a list.

STRINGS in Python:

What is a String?

String in python is a sequence of characters.

What type of characters can we have?

- Small alphabets - [a - z]
- Capital Alphabets - [A - Z]
- Numbers - [0 - 9]
- White spaces
- Special characters

Question: Take a string input from the user. Reverse that string.

```
s = "any string"
s[::-1]
```

Question: Take a string input from the user. Return "PALINDROME" if the string is a palindrome, otherwise return "NOT A PALINDROME".

```
def check_palindrome(a):
    return "PALINDROME" if a == a[::-1] else "NOT A PALINDROME"
```

```
def check_palindrome(a):
    a = a.lower()
    return "PALINDROME" if a == a[::-1] else "NOT A PALINDROME"
```

Question: Take a string input from the user.Print the number of uppercase characters in that string.

```
def count_upper_case(a):
    count = 0
    for i in a:
        if i.isupper():
            count = count + 1
    return count
```

Question: Given a string of comma separated values. Convert it into a string of individual values.

Input:
"56,78,99,101"

Output:
[56, 78, 99, 101]

```
def easy_way(a):
    list_a = a.split(",")
    result = []
    for i in list_a:
        result.append(int(i))
    return result
```

- "".join() --> Input would be a list of strings --> Output would be a string.
- "Random_String".split("_") --> Argument is the splitting character --> Result would be a list.

String Methods:

- The str.split() function converts a string into a list of characters based on a splitting criteria.

```
"4 5 6 7 8 9".split()

['4', '5', '6', '7', '8', '9']
```

```
"This_is_a_underscore_separated_string".split("_")

['This', 'is', 'a', 'underscore', 'separated', 'string']
```

- .join() Opposite of splitting is joining. We can use str.join() to do so.

```
random = "This_is_a_underscore_separated_string".split("_")
" ".join(random)
```

This is a underscore separated string

```
"-".join(random)
```

This-is-a-underscore-separated-string

```
.find()
```

- The .find() method can be used to find a substring inside a string that we can call it on.
- This function returns the starting index of the first occurrence of the substring that we are trying to find inside the string.

```
"this is a random string".find("is")

2
```

```
"this is a random string".find(" a ")

7
```

Returns -1 if the substring is not present.

.replace()

- The method .replace() can be used to replace a substring from the original string with the input we provide.
- This method takes in two arguments, first the substring from the original string and second the string we want to replace it with.
- It replaces the string we provide and returns a new string.

```
random = "This is a random string that I have created"

# The original string is not updated.
random.replace("random", "SUPER RANDOM")

This is a SUPER RANDOM string that I have created
```

.count()

- The method .count() can be used to count the number of occurrences of a substring we provide inside a particular string.

```
random = "This is a random string that I have created"

random.count("a")

5
```

```
random = "This is a random string that I have created random random random"

random.count("random")

4
```

.isdigit()

- .isdigit() returns True a character inside a string is a digit.
- This method works only for integers.

```
"6".isdigit()
True

"a".isdigit()
False

"aasda231321".isdigit()
False

"124143222".isdigit()
True
```

isalpha() -> returns True if a certain string only contains alphabets

```
"aasda231321".isalpha()
False

"a".isalpha()
True

"abc".isalpha()
```

```
True
```

isupper() and islower()

```
"A".isupper()
True

"a".isupper()
False

"a".lower()
a

"A".lower()
a
```

.isspace()

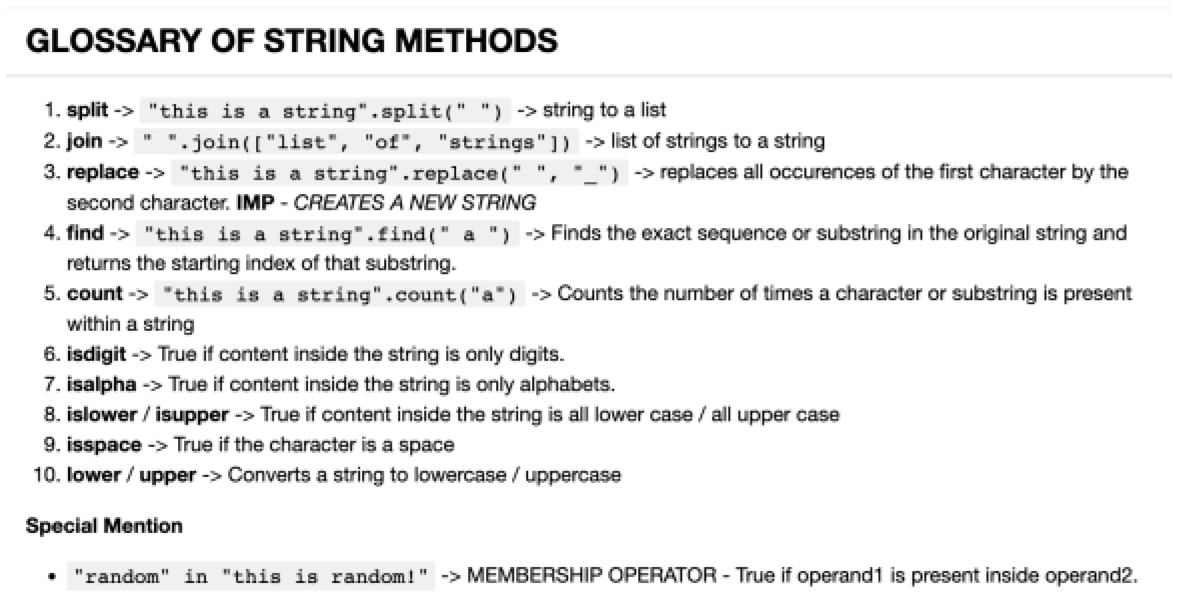
- Returns True if there is a space in the string.

```
"abc".isspace()
False

" ".isspace()
True
```

Question: Take a string as input. Replace all the space with underscore.

```
string = "this is a random string"

#1st Way
def change_to_underscore(s):
    result = ""
    for i in s:
        if i.isspace():
            result = result + "_"
        else:
            result = result + i
    return result

change_to_underscore(string)
```

```
#2nd Way :
string.replace(" ", "_")
```



GLOSSARY OF STRING METHODS

1. **split** -> `"this is a string".split(" ")` -> string to a list
2. **join** -> `" ".join(["list", "of", "strings"])` -> list of strings to a string
3. **replace** -> `"this is a string".replace(" ", "_")` -> replaces all occurences of the first character by the second character. **IMP** - *CREATES A NEW STRING*
4. **find** -> `"this is a string".find(" a ")` -> Finds the exact sequence or substring in the original string and returns the starting index of that substring.
5. **count** -> `"this is a string".count("a")` -> Counts the number of times a character or substring is present within a string
6. **isdigit** -> True if content inside the string is only digits.
7. **isalpha** -> True if content inside the string is only alphabets.
8. **islower** / **isupper** -> True if content inside the string is all lower case / all upper case
9. **isspace** -> True if the character is a space
10. **lower** / **upper** -> Converts a string to lowercase / uppercase

Special Mention

- `"random" in "this is random!"` -> MEMBERSHIP OPERATOR - True if operand1 is present inside operand2.

TUPLES & SETS:

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is **ordered and unchangeable**.

```
# Can change values of elements in a list
a = [1, 2, 3, 4, 5]
a[0] = 10
a

[10, 2, 3, 4, 5]
```

```
# Cannot change values
a = (1, 2, 3, 4, 5)
a[0] = 10
a

TypeError                                 Traceback (most recent call last)
<ipython-input-3-38f47f4ea3ce> in <module>
      1 # Cannot change values
      2 a = (1, 2, 3, 4, 5)
----> 3 a[0] = 10
      4 a

TypeError: 'tuple' object does not support item assignment
```

- The notation to create tuples is the same as lists but instead of square brackets we use parentheses.
- We can do positive indexing, negative indexing on tuples.
- We can even slice tuples.

```
a = (1, 2, 3, 4, 5, 6, 7, 8, 9, 0)
a[2:8]

(3, 4, 5, 6, 7, 8)
```

```
a[-2:-8]

()
```

```
# This will be a tuple
t = ()
type(t)

tuple
```

```
# This will be a tuple
t = (1, 2, 3, 4)
type(t)

tuple
```

```
# This will not be a tuple
t = (45)
type(t)

tuple
```

```
a = [1, 2, 3, 4, 5]
b = tuple(a)
print(a) # List
print(b) # Tuple

[1, 2, 3, 4, 5]
(1, 2, 3, 4, 5)
```

Packing and unpacking in tuples:

```
a = (1, 2, 3)
# We are allowed to do the following
x, y, z = a

print(x) -> 1
print(y) -> 2
print(z) -> 3

# Error
x, y = (1, 2, 3)

ValueError                                Traceback (most recent call last)
<ipython-input-14-d748d0b0a859> in <module>
      1 # Error
----> 2 x, y = (1, 2, 3)

ValueError: too many values to unpack (expected 2)

# Error
x, y, z, m = (1, 2, 3)

ValueError                                Traceback (most recent call last)
<ipython-input-15-585d41b5f86e> in <module>
      1 # Error
----> 2 x, y, z, m = (1, 2, 3)

ValueError: not enough values to unpack (expected 4, got 3)
```

```
# Python uses tuples in it's internal implementation
def foo():
    return 200, 300, 400

bar = foo()
print(bar) # Tuple

(200, 300, 400)
```

```
a, b, c = foo()
print(a)
print(b)
print(c)

200
300
```

```
400
```

Iterating over a list of tuples:

```
a = [(1, "Bipin"), (2, "Ranjan"), (3, "Shanoor"), (4, "Shital"), (5,
"Tharoor"), ]
# prints all the tuples
for i in a:
    print(i)

(1, 'Bipin')
(2, 'Ranjan')
(3, 'Shanoor')
(4, 'Shital')
(5, 'Tharoor')

# We can also do something like
for roll_no, name in a:
    print(f"Roll Number - {roll_no} is {name}!")

Roll Number - 1 is Bipin!
Roll Number - 2 is Ranjan!
Roll Number - 3 is Shanoor!
Roll Number - 4 is Shital!
Roll Number - 5 is Tharoor!
```

Sets:

- Sets are a unique collection of elements.
- We represent a set using curly braces { }
- Sets can store strings, tuples, booleans inside them all at once.
- Sets cannot store Lists, Sets and dictionaries within them.

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

{1, 2, 3, 4, 5, 7}
```

* Sets are not ordered in python. It is a random collection of unique elements.

```
# Error: Because there is no order.
s[0]

TypeError                 Traceback (most recent call last)
<ipython-input-33-c9c96910e542> in <module>
----> 1 s[0]

TypeError: 'set' object is not subscriptable
```

To create an empty set in python we have to use the set() function.

```
s = set()
s

set()
```

```
s = {1, 2, 3}
s

{1, 2, 3}

s.add(4)
s

{1, 2, 3, 4}

s.remove(2)
s

{1,3,4}
```

s.pop() -> The pop() method removes a random item from the set. This method returns the removed item.

Question: Take a string as input. Print the number of unique characters in the string.

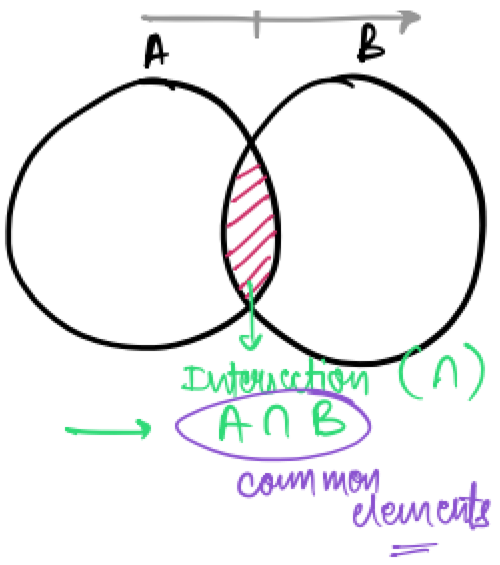
```
def unique_chars(string):
    return len(set(string))

unique_chars("This is a sample sentence")
```

Set Operations:

There are many mathematical operations we can do on sets.

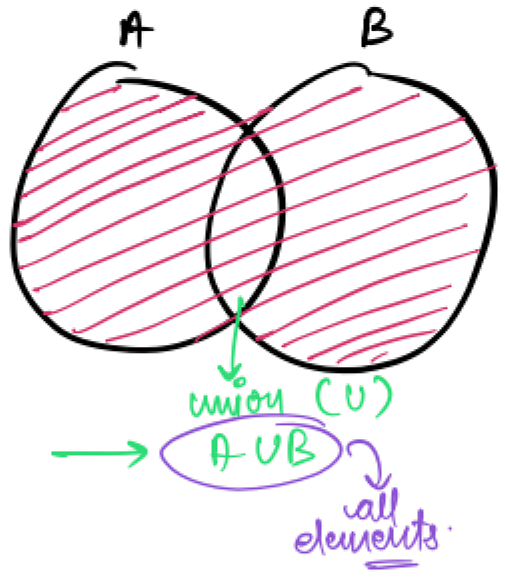
Intersection:

- We can intersect two sets.
- Only contains the common elements in both sets.



```
s1 = {2, 3, 4, 5}
s2 = {1, 3, 4, 6, 7, 8}

s1.intersection(s2)
{3, 4}

s1 & s2
{3, 4}
```

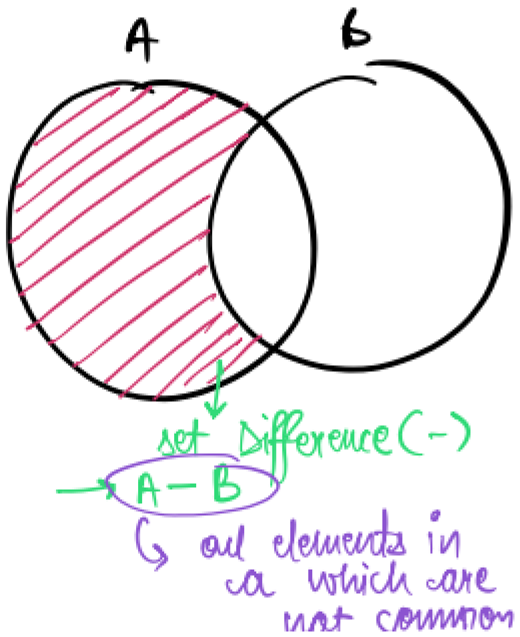
Union: Contains all the elements



```
s1 = {2, 3, 4, 5}
s2 = {1, 3, 4, 6, 7, 8}

s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8}

s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8}
```

Difference:

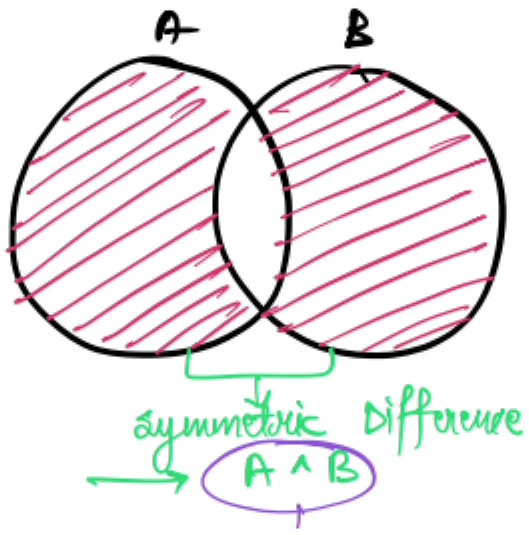
This only contains all elements in A which are not common in B.



```
s1 = {2, 3, 4, 5}
s2 = {1, 3, 4, 6, 7, 8}

s1.difference(s2)
{2, 5}

s1 - s2
{2, 5}
```

Symmetric Difference:

Contains all the elements in union minus the common elements.



```
s1 = {2, 3, 4, 5}
s2 = {1, 3, 4, 6, 7, 8}

s1.symmetric_difference(s2)
{1, 2, 5, 6, 7, 8}

s1 ^ s2
{1, 2, 5, 6, 7, 8}
```

Question: You are given a string as input. Ascertain whether that string contains a binary number or not.

Sample input-1
"0011010101ab"

Sample input-2
"010010101010"

Sample output-1
"Not Binary!"

Sample output-2
"Binary!"

```
# 1
def check_binary(string):
    for i in string:
        if not (i == "0" or i == "1"):
            return "Not Binary!"
    return "Binary!"

print(check_binary("adasfafasfas"))
print(check_binary("01010101010"))

Not Binary!
Binary!
```

```
# 2
def check_binary_cool(string):
    s = set(string)
    if s == {"0", "1"}:
        return "Binary!"
    return "Not Binary!"

print(check_binary_cool("adasfafasfas"))
print(check_binary_cool("01010101010"))

Not Binary!
Binary!
```

DICTIONARY :

- Dictionaries in languages can be used to store the meaning of different words.
- Python uses a syntax which is like a key-value pair with curly braces { } around the elements.

Creating dictionaries in python:

- Curly braces around the values
- Each value has two components: a key and a value.

```
a = {
    "random": "something which is not well defined!"
    "bizzare": "Ajeeb",
     "ship": "Sea transportation"
}
```

Why dictionaries?

- Let's say you are a lead data analyst at some firm.
- You want to store information regarding specific states.
- You can do something like the following.

```
state_wise_data = {
    "Delhi": 450,
    "Haryana": 400,
    "UP": 700
}
```

- Dictionaries are not ordered or subscriptable.
- You access a value directly by using the key.

```
state_wise_data["Haryana"]
400
```

```
a = {}
print(type(a)) # Dict
```

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

```
a = {
    "Delhi": 450,
    "Haryana": 400,
    "UP": 700
}

a["Delhi"] = 500

a # The value is updated
{'Delhi': 500, 'Haryana': 400, 'UP': 700}

a["delhi"] = 800

# Creates a new key
a
{'Delhi': 500, 'Haryana': 400, 'UP': 700, 'delhi': 800}
```

```
avenger = {
    "name": "Thor",
    "age": 1500,
    "weapon": ["mjolnir", "strombreaker"]
}
```

We can also use a method called .update() to update the values.

```
# Old keys will be updated
# New keys will be added
avenger.update(
    {"name": "Thor odinson",
     "weapon": ["strombreaker"],
     "strongest": True,
     "eyes": 1}
)

avenger

{'name': 'Thor odinson',
 'age': 1500,
 'weapon': ['strombreaker'],
 'weapons': ['strombreaker'],
 'strongest': True,
 'eyes': 1}
```

```
random = {
    "a": 1,
    "b": 2,
    "c": 3
}

# Keyerror
random["d"]

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-5d644a66c093> in <module>
      1 # Keyerror
----> 2 random["d"]

KeyError: 'd'
```

```
result = random.get("d")
print(result)

None
```

```
result = random.get("d", "Not Found") # "Not Found" is the default value
print(result)

Not Found
```

How do we remove a key?

```
random = {
    "a": 1,
    "b": 2,
    "c": 3
}
```

```
random.pop("b")
2

random # b is gone

{'a': 1, 'c': 3}
```

We need at least 1 argument when using .pop() on dictionaries.

```
# Throws an error
random.pop()

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-27-2fd5921fdf48> in <module>
      1 # Throws an erorr
----> 2 random.pop()

TypeError: pop expected at least 1 argument, got 0
```

Iterating over dictionaries :

```
a = {"name": "Thor odinson",
     "weapon": ["strombreaker"],
     "strongest": True,
     "eyes": 1}
```

The iterable gets all the keys as values.

```
for i in a:
    print(i)

name
weapon
strongest
eyes
```

To get the values we can do something as follows:

```
for i in a:
    print(a[i])

Thor odinson
['strombreaker']
True
1
```

```
for i in a:
    print(f"{i} -> {a[i]}")

name -> Thor odinson
weapon -> ['strombreaker']
strongest -> True
eyes -> 1
```

.keys() method in dictionaries can be used to get a list of all the keys in the dictionaries.

```
a.keys()

dict_keys(['name', 'weapon', 'strongest', 'eyes'])
```

.values() method in dicionaries can be used to ge a list of all the values in the dictionaries.

```
a.values()

dict_values(['Thor odinson', ['strombreaker'], True, 1])
```

.items() method in dicionaries can be used to ge a list of all the key, value pair in the dictionaries.

```
a.items()

dict_items([('name', 'Thor odinson'), ('weapon', ['strombreaker']),
('strongest', True), ('eyes', 1)])
```

```
# Much cleaner way
for key, value in a.items():
    print(f"{key} -> {value}")

name -> Thor odinson
weapon -> ['strombreaker']
strongest -> True
eyes -> 1
```

How do we check if a key exists within a dictionary?

We can simply use the **in** membership operator to achieve this.

```
"eyes" in a.keys()
True
```

```
"asdasd" in a.keys()
False
```

Can we have an object of any datatype as a key in a dictionary?

- While discussing sets, we talked about how we cannot have sets, dictionaries and lists as elements of sets because they are mutable.
- This is the exact same case with dictionaries.
- Values - Can store any data structure or any data type.
- Keys - Lists, Sets and Dictionaries are not allowed.

```
# Allowed.
random = {
    "key1": [45, 56, 78],
    "key2": {
        "key3": (1, 2, 3)
    },
    45: "value3",
    56.78: "678",
    (1,2,3): "random value"
}
```

```
# Not allowed
```

```
random = {
    "key1": [45, 56, 78],
    "key2": {
        "key3": (1, 2, 3)
    },
    45: "value3",
    56.78: "678",
    [1,2,3]: "random value"
}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-41-a5e17c96aade> in <module>
      1 # Not allowed
----> 2 random = {
      3     "key1": [45, 56, 78],
      4     "key2": {
      5         "key3": (1, 2, 3)

TypeError: unhashable type: 'list'
```

Question: Take a string as input. Create a dictionary according to the following criteria :-

- There will be one key value pair for each unique character
- Key will be the character name and the value will be the count of the character inside the string.

Example input: “rrsssstttt”

Example output:

```
{
  "r": 2,
  "s": 3,
  "t": 4
}
```

```
result = {}
for i in set(string):
    result[i] = string.count(i)
result
```