Course Agenda

- Introduction to Python Programming
- Programming Basics - variables, operators, decision making, iteration, sequences
- Data Structures in Python - list, tuple, set, dictionary
- Functions - Functiona Arguments, lambda functions, function objects, map - filter - reduce
- Exception Handling
- Object Oriented Programming
- Regular Expressions
- Connecting to external data sources
- requests library

Introduction to Python Programming

- Data Types in Python
- Variables in Python
- Data Type conversion
- Accept inputs from user
- Operators in Python

Python is –

- Open source
- Interpreter based
- Platform independent
- Current version – 3.13.x
- Download it from here - https://www.python.org/download/releases/3.0/

In [1]:
```
!python --version
```

```
Python 3.12.4
```

Python Data Types

1. int: Integer numbers, e.g., 42, -7
2. float: Floating-point numbers (decimal), e.g., 3.14, -0.001
3. complex: Complex numbers, e.g., 1+2j, -3+4j
4. bool: Boolean values, either True or False
5. str: String, a sequence of characters, e.g., "hello", 'world'
6. bytes: Immutable sequence of bytes, e.g., b'hello'

In [2]:
```
10 # int
```

Out[2]:
```
10
```

In [3]:
```
15.5 # float
```

Out[3]:
```
15.5
```

In [4]:
```
"abcd" # str
```

Out[4]:
```
'abcd'
```

In [5]:
```
True # bool
```

Out[5]:
```
True
```

In [6]:
```
complex(2,3) # complex
```

Out[6]:
```
(2+3j)
```

Note - Everything in python is an object

Python Containers or Data structures

Containers are any object that holds an arbitrary number of other objects. Generally, containers provide a way to access the contained objects and to iterate over them. Examples of built-in containers include tuples, lists, sets, dictionary.

1. **`List`**
   - Definition: An ordered, mutable (changeable) collection of items.
   - Syntax: Created using square brackets [].
2. **`Tuple`**
   - Definition: An ordered, immutable (unchangeable) collection of items.
   - Syntax: Created using parentheses ().
3. **`Set`**
   - Definition: An unordered collection of unique items.
   - Syntax: Created using curly braces {} or the set() function.
4. **`Dictionary`**
   - Definition: An unordered collection of key-value pairs. Keys must be unique and immutable.
   - Syntax: Created using curly braces {} with key-value pairs separated by colons :.

In [7]:
```
[1, 2, 3, 4, 'abc'] # list
```

Out[7]:
```
[1, 2, 3, 4, 'abc']
```

In [8]:
```
(1, 2, 3, 4) # tuple
```

Out[8]: (1, 2, 3, 4)

In [9]:
```
{2, 's', 3, 5, 5} # set
```

Out[9]: {2, 3, 5, 's'}

In [10]:
```
{1 : "Jane", 2:"George", 3:"Sam"} # dictionary
```

Out[10]: {1: 'Jane', 2: 'George', 3: 'Sam'}

Variables in Python

- A Python variable points to a reserved memory location
- Data or objects are stored in these memory locations
- Variables can be declared by any name or even alphabets

Ex. Define variable name and assign value to the variable

In [13]:
```
name = "Jane"
```

In [16]:
```
print("Welcome", name)
```

```
Welcome 12
```

In [15]:
```
name = 12
```

Note - Python is dynamically typed

`type()` - returns class type of the argument(object) passed as parameter

In [17]:
```
a = True
print(type(a))
```

```
<class 'bool'>
```

In [18]:
```
a = 10
print(type(a))
```

```
<class 'int'>
```

In [19]:
```
a = 10.5
print(type(a))
```

```
<class 'float'>
```

In [20]:
```
a = "abcd"
print(type(a))
```

```
<class 'str'>
```

In [21]:
```
lst = [1, 2, 3, 4]
print(type(lst))
```

```
<class 'list'>
```

Ex. WAP to take name of user as input and print a welcome message

In [22]:
```
name = input("Enter your name - ")
print("Welcome", name)
```

```
Welcome George
```

Ex. WAP to take two numbers as input from user and print their sum.

In [29]:
```
num1 = int(input("Enter a number - "))
num2 = int(input("Enter a number - "))

print(num1 + num2)
```

```
12
```

In [3]:
```
type(num1)
```

Out[3]: str

Note - input() always take values in str format

Data Type Conversion

`Implicit Conversion:` Conversion done by Python interpreter without programmer's intervention

In [4]:
```
a = 10  # int
b = 2.5 # float
a + b
```

Out[4]: 12.5

`Explicit Conversion:` Conversion that is user-defined that forces an expression to be of specific data type

In [5]:
```
a = "10"  # str
b = 5     # int

int(a) + b
```

Out[5]: 15

Common Type Casting Functions:

- int(): Converts a value to an integer.
- float(): Converts a value to a floating-point number.
- str(): Converts a value to a string.
- bool(): Converts a value to it bool equivalent
- list(): Converts a value to a list.

- tuple(): Converts a value to a tuple.
- set(): Converts a value to a set.
- dict(): Converts a sequence of key-value pairs into a dictionary.

`int()` conversion

In [6]:
```
x = 10.8 # float
int(x)
```

Out[6]:
```
10
```

In [7]:
```
x = "10" # int value in str format
int(x)
```

Out[7]:
```
10
```

In [8]:
```
x = True # bool
int(x)
```

Out[8]:
```
1
```

In [9]:
```
x = False # bool
int(x)
```

Out[9]:
```
0
```

In [10]:
```
x = "abcd" # str
int(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 2
      1 x = "abcd" # str
----> 2 int(x)

ValueError: invalid literal for int() with base 10: 'abcd'
```

In [11]:
```
x = "10.8" # float value in str format
int(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[11], line 2
      1 x = "10.8" # float value in str format
----> 2 int(x)

ValueError: invalid literal for int() with base 10: '10.8'
```

`float()` conversion

In [12]:
```
x = True # bool
float(x)
```

Out[12]: 1.0

In [13]:
```
x = False # bool
float(x)
```

Out[13]: 0.0

In [14]:
```
x = 10 # int
float(x)
```

Out[14]: 10.0

In [15]:
```
x = "10.8" # float value in str format
float(x)
```

Out[15]: 10.8

In [16]:
```
x = "abcd" # str
float(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 2
      1 x = "abcd" # str
----> 2 float(x)

ValueError: could not convert string to float: 'abcd'
```

`str()` conversion

In [17]:
```
x = 10
str(x)
```

Out[17]: '10'

In [18]:
```
lst = [1,2,3,4,5]
str(lst)
```

Out[18]: '[1, 2, 3, 4, 5]'

`bool()` conversion

In [19]:
```
x = 0
bool(x)
```

Out[19]: False

In [20]:
```
x = 1
bool(x)
```

Out[20]: True

In [21]:
```
x = 0.0
bool(x)
```

Out[21]: False

In [22]:
```
x = 1.0
bool(x)
```

Out[22]: True

In [23]:
```
x = 10
bool(x)
```

Out[23]: True

In [25]:
```
x = "abc"
bool(x)
```

Out[25]: True

In [26]:
```
x = "True"
bool(x)
```

Out[26]: True

In [27]:
```
x = "False"
bool(x)
```

Out[27]: True

In [28]:
```
x = None
bool(x)
```

Out[28]: False

Note - bool() returns True for any value except 0 or 0.0 or None

---

Operators in Python

Operators are special symbols in Python that carry out computations. The value that the operator operates on is called as operand.

1. Arithmetic Operators

These operators perform arithmetic operations on numeric values.

- `+` : Addition
- `-` : Subtraction

- `*` : Multiplication
- `/` : Division
- `%` : Modulus (remainder of division)
- `**` : Exponentiation (power)
- `//` : Floor division (division that results in the largest integer less than or equal to the quotient)

In [30]:
```
a = 10
b = 6
```

In [31]:
```
print("Addition - ", a+b)
```

```
Addition -  16
```

In [32]:
```
print("Substraction - ", a-b)
```

```
Substraction -  4
```

In [33]:
```
print("Multiplication - ", a*b)
```

```
Multiplication -  60
```

In [34]:
```
print("Division - ", a/b) # returns result in float format
```

```
Division -  1.6666666666666667
```

In [35]:
```
print("Floor Division - ", a//b) # returns integer part of the division
```

```
Floor Division -  1
```

In [36]:
```
print("Exponential - ", a**b) # returns the result of a to the power b
```

```
Exponential -  1000000
```

In [37]:
```
print("Modulous - ", a%b) # returns remainder of the division
```

```
Modulous -  4
```

Ex. WAP to calculate BMI of a person.

In [43]:
```
weight = int(input("Enter your weight kgs - "))
height = float(input("Enter your height in mtrs - "))
bmi = round(weight/(height ** 2), 2)
print(bmi)
```

```
21.26
```

Ex. WAP to accept hours and rate per hour from user and compute gross pay.

In [38]:
```
hrs = int(input("Enter number of hrs - "))
rate = int(input("Enter rate per hr - "))
gross_pay = rate * hrs

print(gross_pay)
```

```
50000
```

2. Comparison Operators

These operators compare two values and return a boolean result (True or False).

- `==` : Equal to
- `!=` : Not equal to
- `>` : Greater than
- `< : Less than
- `>=` : Greater than or equal to
- `<=` : Less than or equal to

In [44]:
```
a = 10
b = 7
```

In [45]:
```
print(a < b)
```

```
False
```

In [46]:
```
print(a <= b)
```

```
False
```

In [47]:
```
print(a > b)
```

```
True
```

In [48]:
```
print(a >= b)
```

```
True
```

In [49]:
```
print(a==b)
```

```
False
```

In [50]:
```
print(a!=b)
```

```
True
```

Ex. WAP to take a number as input and write condition to check if the number is greater than 10.. (Output must be a bool value)

In [ ]:
```
num = int(input("Enter a number - "))
```

Ex. WAP to take a number as input and write condition to check if the number is divisible by 5. (Output must be a bool value)

In [ ]:
```
num = int(input("Enter a number - "))
```

3. Logical Operators

These operators are used to combine conditional statements.

- `and` : Returns True if both statements are true
- `or` : Returns True if at least one of the statements is true

- `not` : Reverses the result, returns False if the result is true

Ex. WAP to check if the number is divisible by 5 and greater than 10

In [52]:
```
num = int(input("Enter a number - "))
num % 5 == 0 and num > 10
```

Out[52]: True

Ex. WAP to check if the number is either divisible by 5 or greater than 10 or both

In [53]:
```
num = int(input("Enter a number - "))
num % 5 == 0 or num > 10
```

Out[53]: True

In [54]:
```
not True
```

Out[54]: False

In [55]:
```
not False
```

Out[55]: True

In [56]:
```
not(10 - 5 * 2)
```

Out[56]: True

4. Basic Assignment Operator

`=` : Assigns the value on the right to the variable on the left.

Compound Assignment Operators

These operators perform an operation on a variable and then assign the result back to that variable.

- `+=` : Adds the right operand to the left operand and assigns the result to the left operand.
- `-=` : Subtracts the right operand from the left operand and assigns the result to the left operand.
- `*=` : Multiplies the left operand by the right operand and assigns the result to the left operand.
- `/=` : Divides the left operand by the right operand and assigns the result to the left operand.
- `%=` : Takes the modulus of the left operand by the right operand and assigns the result to the left operand.
- `//=` : Performs floor division on the left operand by the right operand and assigns the result to the left operand.

- `**=` : Raises the left operand to the power of the right operand and assigns the result to the left operand.

In [ ]:
```
var = 10 - 5 * 4
```

In [ ]:
```
a = 10
a = a + (10 - 5 * 4)
a += (10 - 5 * 4)
```

In [ ]:
```
count += 10
```

Membership Operators

Membership operators checks whether a value is a member of a sequence.

Sequence Object -

A `sequence` is defined as a collection of arbitrary number of elements. The sequence may be a list, a string, a tuple, or a dictionary

- in - The `in` operator is used to check if a value exists in any sequence object or not.
- not in - A `not in` works in an opposite way to an 'in' operator. A 'not in' evaluates to True if a value is not found in the specified sequence object. Else it returns a False.

In [57]:
```
"a" in "abcd"
```

Out[57]: True

In [58]:
```
"a" not in "abcd"  # str is seq of characters
```

Out[58]: False

Ex. WAP to check if entered name is present in the mentioned list or not

In [59]:
```
names = ["Jane", "George", "Sam"]
"Jack" in names
```

Out[59]: False

In [60]:
```
names = ["Jane", "George", "Sam"]  # list is sequence of objects
"J" in names
```

Out[60]: False

Ex. WAP to check if entered character is a vowel or not

In [62]:
```
ch = input("Enter a character - ")
ch in "aeiou"
```

Out[62]: False

In [63]:
```
"abc"
```

Out[63]:
```
'abc'
```

In [64]:
```
'abc'
```

Out[64]:
```
'abc'
```

Ex. WAP to calculate the hypoteneous of a right angled triangle when sides are given

In [67]:
```
import math as m
base = 4
height = 3

hypt = m.sqrt((base ** 2) + (height**2))
hypt
```

Out[67]:
```
5.0
```

Working with Modules

In [68]:
```
import math
math.sqrt(64)
```

Out[68]:
```
8.0
```

In [68]:
```
import math as m
m.sqrt(64)
```

Out[68]:
```
8.0
```

In [78]:
```
from math import sqrt
sqrt(64)

# drawback - can conflict with exsisting functions or variables
```

Out[78]:
```
8.0
```

In [79]:
```
type(sqrt)
```

Out[79]:
```
builtin_function_or_method
```

In [83]:
```
from math import sqrt
print(sqrt(64))

sqrt = 10

base = 4
height = 3

hypt = sqrt((base ** 2) + (height**2))
```

```
hypt
```

```
8.0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[83], line 9
      6 base = 4
      7 height = 3
----> 9 hypt = sqrt((base ** 2) + (height**2))
     11 hypt

TypeError: 'int' object is not callable
```

In [84]:
```
type(sqrt)
```

Out[84]:
```
int
```

In [80]:
```
x = 10
```

In [81]:
```
x = x + 5
```

In [86]:
```
print(dir(__builtins__))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BaseExcept
ionGroup', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'Chi
ldProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedErro
r', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EncodingW
arning', 'EnvironmentError', 'Exception', 'ExceptionGroup', 'False', 'FileExistsErro
r', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IO
Error', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'Interrupt
edError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'Memo
ryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImpl
emented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarn
ing', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError',
'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIter
ation', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'Ti
meoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'Unicod
eEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarni
ng', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__IPYTHON__', '_
_build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__
package__', '__spec__', 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'boo
l', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile',
'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'e
numerate', 'eval', 'exec', 'execfile', 'filter', 'float', 'format', 'frozenset', 'ge
t_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input',
'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'ma
p', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'prin
t', 'property', 'range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr',
'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars',
'zip']
```

In [88]:
```
help(len)  # prints the documentation of the function
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

In [90]:
```
import sys
print(sys.builtin_module_names)
```

```
('_abc', '_ast', '_bisect', '_blake2', '_codecs', '_codecs_cn', '_codecs_hk', '_code
cs_iso2022', '_codecs_jp', '_codecs_kr', '_codecs_tw', '_collections', '_contextvar
s', '_csv', '_datetime', '_functools', '_heapq', '_imp', '_io', '_json', '_locale',
'_lsprof', '_md5', '_multibytecodec', '_opcode', '_operator', '_pickle', '_random',
'_sha1', '_sha2', '_sha3', '_signal', '_sre', '_stat', '_statistics', '_string', '_s
truct', '_symtable', '_thread', '_tokenize', '_tracemalloc', '_typing', '_warnings',
'_weakref', '_winapi', '_xxinterpchannels', '_xxsubinterpreters', 'array', 'atexit',
'audioop', 'binascii', 'builtins', 'cmath', 'errno', 'faulthandler', 'gc', 'itertool
s', 'marshal', 'math', 'mmap', 'msvcrt', 'nt', 'sys', 'time', 'winreg', 'xxsubtype',
'zlib')
```

Note - To create your custom module

1. Create a python file as file.py
2. Define all the functions/variable/Classes
3. Save the file in same folder where you want to import the module
4. In new file import the custom module as "import file" or "import file as f"

Decision Making

Decision-making statements in Python allow you to control the flow of execution based on certain conditions. These statements include if, elif, and else and can be used to execute different blocks of code depending on whether conditions are True or False.

- ## Basic if Statement
  - The if statement evaluates a condition and executes a block of code if the condition is True.
  - Syntax:
    `if condition:`
        `# block of code`
- ## if-else Statement
  - The if-else statement evaluates a condition and executes one block of code if the condition is True, and another block if the condition is False.
  - Syntax:
    `if condition:`
        `# block of code if condition is True`

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

- ## if-elif-else Statement
  - The if-elif-else statement allows you to check multiple expressions for True and execute a block of code as soon as one of the conditions evaluates to True.
  - If none of the conditions are True, the else block is executed.
  - Syntax:

```
if condition1:
    # block of code if condition1 is True
elif condition2:
    # block of code if condition2 is True
elif condition3:
    # block of code if condition3 is True
else:
    # block of code if none of the conditions are True
```

- ## Nested if Statements
  - You can nest if statements within other if statements to check multiple conditions in a hierarchical manner.
  - Syntax:

```
if condition1:
    # block of code if condition1 is True
    if condition2:
        # block of code if condition2 is True
    else condition3:
        # block of code if condition2 is False
else:
    # block of code if condition1 is False
```

- ## One-Line if-else
  - Syntax:

```
value_if_true if condition else value_if_false
```

Examples -

Ex. WAP to take a character as input and print if it is a vowel

In [92]:
```
ch = input("Enter a character - ")
if ch in "aeiou" :
    print("Vowel")
```

Ex. WAP to take an input if it is a single character and a vowel, else print invalid

In [104…
```
ch = input("Enter a character - ").lower()
if ch in ['a', 'e', 'i', 'o', 'u'] :
    print("Vowel")
else:
    print("Invalid")
```

```
Vowel
```

Ex. WAP to take an input check if it is a single character and a vowel or consonant, else print invalid

In [111…
```
ch = input("Enter a character - ").lower()
if len(ch) == 1 :
    if ch in "aeiou" :
        print("Vowel")
    else:
        print("Consonant")
else:
    print("Invalid")
```

```
Consonant
```

Ex. WAP to take an input check if it is a single character and a vowel or consonant, else print invalid with custom message

In [116…
```
ch = input("Enter a character - ").lower()
if len(ch) != 1 :
    print("len must be 1.")
elif not ch.isalpha():
    print("input must be a str.")
else :
    if ch in "aeiou" :
        print("Vowel")
    else:
        print("Consonant")
```

```
Consonant
```

Ex. WAP to accept a single character from user and check if it is a vowel or not.

In [ ]:
```

```

Ex. WAP to take weight in kgs and height in mtrs from user. Calculate BMI and print if the the health status based on following chart

BMI Categories

- Underweight - BMI < 18.5
- Normal weight - BMI between 18.5-24.9
- Overweight - BMI between 25-29.9
- Obesity - BMI > 30

In [ ]:
```

```

Ex. WAP to accept hours and rate per hour from user and compute gross pay.

- for 40 hrs pay the standard rate

- if overtime then pay 1.5 times of rate for the additional hrs.

In [118…
```
hrs = int(input("Enter number of hrs - "))
rate = int(input("Enter rate per hr - "))

if hrs <= 40 :
    gross_pay = rate * hrs
else:
    gross_pay = (40 * rate) + ((hrs - 40) * 1.5 * rate)

print(gross_pay)
```

```
47500.0
```

Ex. Toss a coin and guess the outcome

WAP to simulate coin toss and compare the outcome with guess made by user. The program must return "Invalid input" if user enters a wrong or invalid value as a guess.

Algorithm

1. take guess as heads or tails as input from user
2. check if it is a valid guess
3. generate outcome as heads or tails randomly and print the outcome
4. compare guess and outcome and print result and win or lost

In [131…
```
import random as r
guess = input("Enter your guess as heads or tails - ")
options = ("heads", "tails")
if guess in options :
    outcome = r.choice(options)
    print("Outcome - ", outcome)
    if guess == outcome:
        print("You Win")
    else:
        print("You Lost")
else:
    print("Invalid Input")
```

```
Outcome -  tails
You Win
```

random module in Python

- Generates a random value
- Importing random module
  import random as r
- Frequently used functions
  - **r.random()** – Generates a random float number between 0.0 to 1.0
  - **r.randint()** – Returns a random integer between the specified integers

- **r.randrange()** – Returns a randomly selected element from the range created by the start, stop and step arguments
- **r.choice()** – Returns a randomly selected element from a non-empty sequence
- **r.shuffle()** – This functions randomly reorders the elements in a list

In [124…
```
import random as r # as is an operator - used to assign a variable to a moudule whi
```

In [125…
```
r.random() # generates a random value between 0 and 1
```

Out[125…
```
0.24003709575789056
```

In [126…
```
r.randint(1,5) # generates a random integer between the mentioned start and end val
```

Out[126…
```
5
```

In [127…
```
r.choice([1,2,3,4]) # generates a random value from the mentioned iterable or seque
```

Out[127…
```
1
```

In [128…
```
r.choice(["abc", "pqr", "xyz"])
```

Out[128…
```
'pqr'
```

In [129…
```
r.choice("abcde")
```

Out[129…
```
'e'
```

In [130…
```
help(r.choice)
```

```
Help on method choice in module random:

choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

Ex. 7up and 7down

WAP to simulate the below mentioned scenario -

1. Player enters the game with initial amount as Rs. 1,000/-
2. Generate a random value between 1 to 14 and store it in variable "outcome"
3. if outcome < 7, player looses amount by (outcome*100)
4. if outcome > 7, player earns amount by (outcome*100)
5. if outcome = 7, player hits a jackpot and wins Rs. 1,00,00,000.
6. Print the final amount with the player.

In [134…
```
import random as r
amt = 1000
outcome = r.randint(1, 14)
print("Your Score - ", outcome)
if outcome < 7 :
```

```
    amt -= (outcome * 100)
elif outcome > 7 :
    amt += (outcome * 100)
else:
    amt += 1000000
    print("You have hit the jackpot!!!!")
print("Final Balance - ", amt)
```

```
Your Score -  9
Final Balance -  1900
```

Loops

Loops are used to execute of a specific block of code in repetitively

while loop

- The 'while loop' in Python is used to iterate over a block of code as long as the test expression holds true
- Event based loop
- Event occurring inside the loop determines the number of iterations
- This loop is used when the number of times to iterate is not known to us beforehand

Ex. Modify the 7up 7down program based on following rules -

- Ask user his choice to play again as `yes/no`.
- First round starts with amount balance as Rs. 1000. However, further rounds will be played on the balance amount generated from previous round. Example - in round 1 user earned Rs. 800. So for his next round amount will be Rs. 1,800 which is balance generated in previous round.
- The game will terminate if user -
  - choice to play again is `no`
  - hits the `jackpot`
  - has insufficient funds to play the next round.

In [142…

```
import random as r
amt = 1000
choice = "yes"

while choice == "yes" :
    outcome = r.randint(1, 14)
    print("Your Score - ", outcome)
    if outcome < 7 :
        amt -= (outcome * 100)
    elif outcome > 7 :
        amt += (outcome * 100)
```

```
    else:
        amt += 1000000
        print("You have hit the jackpot!!!!")
        break

    if amt <= 600 :
        print(f"Insufficient Funds - Rs.{amt}. Do you wish to top-up?")
        choice = input("yes/no - ")
        if choice == "yes" :
            amt += 1000
            continue
        break

    print(f"Your current balance is Rs.{amt}. Do you wish to continue?")
    choice = input("yes/no - ")

print("Final Balance - ", amt)
```

```
Your Score -  6
Insufficient Funds - Rs.400. Do you wish to top-up?
Your Score -  5
Your current balance is Rs.900. Do you wish to continue?
Your Score -  12
Your current balance is Rs.2100. Do you wish to continue?
Your Score -  10
Your current balance is Rs.3100. Do you wish to continue?
Final Balance -  3100
```

break statement

- The 'break' statement ends the loop and resumes execution at the next statement
- The break statement can be used in both 'while' loop and 'for' loop
- It is always used with conditional statements

continue statement

- The 'continue' statement in Python ignores all the remaining statements in the iteration of the current loop and moves the control back to the beginning of the loop
- The continue statement can be used in both 'while' loop and 'for' loop
- It is always used with conditional statements

Ex. Write a code to validate user input for an integer input

In [ ]:

```
```

for loop

- The 'for loop' in Python is used to iterate over the items of a sequence object like list, tuple, string and other iterable objects
- The iteration continues until we reach the last item in the sequence object
- Counter driven loop

- This loop is used when the number of times to iterate is predefined

Ex. WAP to print square of numbers in the given list

In [143…
```
lst = [1, 2, 3, 4, 5]
for i in lst :
    print(i, " - ", i**2)
```

```
1  -  1
2  -  4
3  -  9
4  -  16
5  -  25
```

Ex. WAP to print square of all even numbers in the given list

In [144…
```
lst = [1, 2, 3, 4, 5]
for i in lst :
    if i % 2 == 0 :
        print(i, " - ", i**2)
```

```
2  -  4
4  -  16
```

Ex. WAP to accept a word from user and print vowels in the word

In [146…
```
word = input("Enter a word - ")
for i in "aeiou" :
    if i.lower() in word :
        print(i)
```

```
i
```

Ex. Write a python code to print the product of all elements in the numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11, 43]?

In [147…
```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11, 43]
product = 1
for i in numbers :
    product *= i
product
```

Out[147…
```
54489600
```

In [148…
```
import math as m
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11, 43]
m.prod(numbers)
```

Out[148…
```
54489600
```

In [149…
```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11, 43]
sum(numbers)
```

Out[149…
```
91
```

Ex. WAP to perform product of first 10 natural numbers

In [157…
```
m.prod(range(1, 11))
```

Out[157…
```
3628800
```

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 end.
- `step` - Optional. An integer number specifying the incrementation. Default is 1

In [151…
```
range(1, 11)  # 1 - 10 - sequence object of integers
```

Out[151…
```
range(1, 11)
```

In [152…
```
for i in range(1,11):
    print(i, end = ",")
```

```
1,2,3,4,5,6,7,8,9,10,
```

In [153…
```
for i in range(11):
    print(i, end = ",")
```

```
0,1,2,3,4,5,6,7,8,9,10,
```

In [154…
```
for i in range(1, 50, 3):
    print(i, end = ", ")
```

```
1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49,
```

In [155…
```
for i in range(1, 50, 5):
    print(i, end = ", ")
```

```
1, 6, 11, 16, 21, 26, 31, 36, 41, 46,
```

In [156…
```
for i in range(50, 0, -1):
    print(i, end = ", ")
```

```
50, 49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30,
29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9,
8, 7, 6, 5, 4, 3, 2, 1,
```

In [150…
```
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
 |  Methods defined here:
 |
 |  __bool__(self, /)
 |      True if self else False
 |
 |  __contains__(self, key, /)
 |      Return bool(key in self).
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
```

```
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(...)
|      Return a reverse iterator.
|
|  count(...)
|      rangeobject.count(value) -> integer -- return number of occurrences of value
|
|  index(...)
|      rangeobject.index(value) -> integer -- return index of value.
|      Raise ValueError if the value is not present.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs)
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  start
|
|  step
|
|  stop
```

Note -

1. iterate on sequence - for loop
2. DO NOT use indexing to iterate on any sequence
3. Avoid while loop when working on sequence
4. Do not modify iterating variable inside for loop

Examples

Ex. WAP to accept an integer from user and print a table for given number.

3 x 1 = 3

.

.

3 x 10 = 30

In [158…
```
lst
```

Out[158…
```
[1, 2, 3, 4, 5]
```

In [160…
```
for i in range(len(lst)):
    print(lst[i])
```

```
1
2
3
4
5
```

for-else loop

- if break statement is executed then else will not be executed
- if break statement is not executed then else will be executed

In [162…
```
for i in range(10):
    print(i,end = " ")
    if i == 7 :
        print("Terminated")
        break
else:
    print("Else block executed")
```

```
0 1 2 3 4 5 6 7 Terminated
```

In [163…
```
for i in range(6):
    print(i,end = " ")
    if i == 7 :
        print("Terminated")
        break
else:
    print("Else block executed")
```

```
0 1 2 3 4 5 Else block executed
```

Example - prime numbers

In [167…
```
num = 170
for i in range(2, num) :
    if num % i == 0 :  # tells number is not prime
        print("Not Prime")
        break
else:
    print("Prime")
```

```
Not Prime
```

Example on implicit conversion

In [172…
```
num = int(input("Enter a number - "))
if num % 2:
    print("odd")
else:
    print("even")
```

```
even
```

In [175…
```
num = int(input("Enter a number - "))
if num % 5: # divisibility by 5
    print("not divisible by 5")
```

```
else:
    print("divisible by 5")
```

```
divisible by 5
```

Strings in Python

Strings are -

- an ordered sequence of characters
- enclosed in a pair of single quotes or pair of double quotes
- immutable

Empty string

In [168…
```
string = ''

string = ""
```

In [171…
```
word = input("Enter a word - ")
if word :   # check for string as empty?
    print(word)
else :
    print("Empty String")
```

```
Empty String
```

Note - bool() of empty str is always

Defining a string

In [176…
```
string = "aero-plane"
```

Operations on strings

- Indexing
- Slicing
- Concatenation
- Repeatition
- Membership
- Iteration

Indexing in Strings

- Each character in a string has a unique index, starting from 0 for the first character up to n-1 for the last character, where n is the length of the string.

- **Positive Indexing** - Positive indexing starts from 0 and goes up to n-1.
    - Index 0 corresponds to the first character.
    - Index 1 corresponds to the second character, and so on.
- **Negative Indexing** - Negative indexing starts from -1 for the last character and goes up to -n for the first character.
    - Index -1 corresponds to the last character.
    - Index -2 corresponds to the second last character, and so on.
- **Accessing Substrings** - You can also use slicing to access substrings. The syntax for slicing is `string[start:stop:step]`, where:
    - start is the starting index (inclusive).
    - stop is the ending index (exclusive).
    - step is the step size (optional).

Extract first element from the string

In [177…
```
string[0]
```
Out[177…
```
'a'
```

Extract 5th element from the string

In [178…
```
string[4]
```
Out[178…
```
'-'
```

Ex. Extract last element from the string

In [179…
```
string[-1]
```
Out[179…
```
'e'
```

Ex. Extract first 3 characters from string

In [180…
```
string[0:3]
```
Out[180…
```
'aer'
```

Ex. Extract all characters from index position 3

In [181…
```
string[3 : ]
```
Out[181…
```
'o-plane'
```

Ex. Extract last 4 characters from the string

In [182…
```
string[-4 :]
```

Out[182… 'lane'

Reverse of string

In [183…
```
string[::-1]
```

Out[183… 'enalp-orea'

Ex. WAP to check if entered string is a palindrome or not.

`Palindrome` - the string reads same characters left to right or right to left

Ex - madam

In [ ]:

Ex. WAP to generate a new string by swapping first and last characers

ex - abcde - ebcda

In [187…
```
word = input("Enter a word - ")
word = word[-1] + word[1 : -1]+ word[0]
word
```

Out[187… 'eingapors'

`Concatenation` - merging two strings into a single object using the + operator.

In [185…
```
"abc" + "pqr"
```

Out[185… 'abcpqr'

`Repetition` - The repetition operator * will make multiple copies of that particular object and combines them together.

In [186…
```
"abc" * 3
```

Out[186… 'abcabcabc'

Strings are immutable and cannot be modified

In [189…
```
strg = "abcd"
strg[0] = "x"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[189], line 2
      1 strg = "abcd"
----> 2 strg[0] = "x"

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

Built-in Functions

- **len()** - returns length of the string
- **min(), max()** - returns minimum and maximum element from the string
- **sorted()** - sorts the characters of the string and returns a list

In [190…
```
string = "Mississippi"

len(string)
```

Out[190…
```
11
```

In [191…
```
min(string)
```

Out[191…
```
'M'
```

In [192…
```
max(string)
```

Out[192…
```
's'
```

In [193…
```
sorted(string) # returns a list object
```

Out[193…
```
['M', 'i', 'i', 'i', 'i', 'p', 'p', 's', 's', 's', 's']
```

Strings Methods

- **str.index(`obj`)** - returns index of the first occurence of the character
- **str.count(`obj`)** - returns count of number of occurences of the charcter
- **str.upper()** - returns string of uppercase characters
- **str.lower()** - returns string of lowercase characters
- **str.title()** - returns string of sentence case charaters
- **str.isupper()** - checks if all characters are uppercase
- **str.islower()** - checks if all characters are lowercase
- **str.isdigit()** - checks if all characters are digits
- **str.isalpha()** - checks if all characters are alphabets
- **str.isalnum()** - checks if all characters are either alphabets or digits
- **str.split(`delimiter`)** - splits the string on the mentioned delimiter and returns a list of obtained parts of the string

- **str.replace(`str` , `str`)** - replaces all the mentioned characters with the specified string and returns a new string
- **str.strip(`delimiter`)** - removes whitespace characters from start and end of the string (delimiter can also be specified)
- **`delimiter` .join(** `sequence`) - it is called on a string object which acts as a delimiter to join all string elements in the sequence passed as an argument to join()

In [194…
```
print(dir(str))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq_
_', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__
getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__l
e__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '_
_str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalp
ha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isp
rintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'mak
etrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'str
ip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

In [195…
```
strg = "Have a nice day"
strg.split()
```

Out[195… `['Have', 'a', 'nice', 'day']`

In [196…
```
strg = "Have a nice day"
strg.split("a")
```

Out[196… `['H', 've ', ' nice d', 'y']`

In [197…
```
strg.replace("a", "*")
```

Out[197… `'H*ve * nice d*y'`

In [198…
```
lst = ['Have', 'a', 'nice', 'day']
"_".join(lst)
```

Out[198… `'Have_a_nice_day'`

In [199…
```
sales = "   $20000"
sales.strip()
```

Out[199… `'$20000'`

In [200…
```
sales = "   $20000"
sales.strip().strip("$")
```

Out[200… `'20000'`

Examples

Ex. WAP to convert the given string -

string = "I am in Python class"

o/p - 'ssalc nohtyP ni ma I'

o/p - 'I Am In Python Class'

In [203…
```
string = "I am in Python class"
string[::-1]
```

Out[203…
```
'ssalc nohtyP ni ma I'
```

In [204…
```
string.title()
```

Out[204…
```
'I Am In Python Class'
```

Ex. WAP to replace all vowels in a word with and asterisk.

In [207…
```
word = input("Enter a word - ")
for i in "aeiouAEIOU" :
    word = word.replace(i, "*")
word
```

Out[207…
```
's*ng*p*r*'
```

In [212…
```
word = input("Enter a word - ")
trans_obj = str.maketrans("aeiou", "*****")
word.translate(trans_obj)
```

Out[212…
```
's*ng*p*r*'
```

In [211…
```
word = input("Enter a word - ")
trans_obj = str.maketrans("aeiou", "@3!0^")
word.translate(trans_obj)
```

Out[211…
```
's!ng@p0r3'
```

Ex. WAP to convert the given profit value to int

In [214…
```
profit = "($1,200)"  # -1200
trans_obj = str.maketrans("(", "-", "$,)")
int(profit.translate(trans_obj))
```

Out[214…
```
-1200
```

Ex. WAP to print foloowing pattern

In [ ]:
```
*
**
```

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

In [202…
```
for i in range(1, 6) :
    print("* "*i)
```

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

Ex. WAP to accept numbers from user in comma seperated format. Extract the integers and perform their summation

In [ ]:

Ex. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '#', except the first char itself.

Sample String : 'restart'

Expected Result : 'resta#t'

In [ ]:

Tuples in Python

Tuples -

- are Python containers
- are an ordered sequence of mixed data
- enclose elements in a pair of round brackets, separated by commas
- are immutable

Defining a tuple

In [215…
```
tup = (1, 2, 3, 4)
tup
```

Out[215…
```
(1, 2, 3, 4)
```

Empty Tuple

In [216…
```
tup = ()

tup = tuple()
```

Note - Any sequence can be converted into a tuple

In [218…
```
tuple(range(1, 11))
```

Out[218…
```
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Single value Tuple

In [219…
```
(10,)
```

Out[219…
```
(10,)
```

Indexing and Slicing

In [221…
```
fam = ("Rosie", 34, "Sam", 36, "Jonnah", 15, "Jessie", 12)
```

Ex. Extract 1st element from tuple -

In [222…
```
fam[0]
```

Out[222…
```
'Rosie'
```

Ex. Extract last element from tuple -

In [223…
```
fam[-1]
```

Out[223…
```
12
```

Ex. Extract first 3 elements from tuple -

In [224…
```
fam[0:3]
```

Out[224…
```
('Rosie', 34, 'Sam')
```

Ex. Extract last 2 element from tuple -

In [225…
```
fam[-2 :]
```

Out[225…
```
('Jessie', 12)
```

Ex. Extract "True" from the given Tuple

In [226…
```
mix_tuple = (('a', 1, True), 234567, 'Science', -5)
mix_tuple[0][2]
```

Out[226…
```
True
```

In [227…
```
mix_tuple = (('a', 1, True), 234567, 'Science', -5)
mix_tuple[1][2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[227], line 2
      1 mix_tuple = (('a', 1, True), 234567, 'Science', -5)
----> 2 mix_tuple[1][2]

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

In [228…
```
mix_tuple = (('a', 1, True), 234567, 'Science', -5)
mix_tuple[2][2]
```

Out[228…
```
'i'
```

Operations on Tuples

- Iteration
- Membership
- Concatenation
- Repetition

In [ ]:
```
```

`Concatenation` - merging two tuples into a single tuple object using the + operator.

In [229…
```
(1, 2, 3) + (4, 5, 6)
```

Out[229…
```
(1, 2, 3, 4, 5, 6)
```

`Repetition` - The repetition operator * will make multiple copies of that particular object and combines them together.

In [230…
```
(1, 2, 3) * 3
```

Out[230…
```
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Tuples are immutable hence cannot be modified

In [231…
```
tup = (1, 2, 3, 4)
tup[0] = 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[231], line 2
      1 tup = (1, 2, 3, 4)
----> 2 tup[0] = 10

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

Built-in functions on Tuples

- **len()** - returns length of the tuple
- **min(), max()** - returns minimum and maximum element from the tuple
- **sorted()** - sorts the elements of the tuple and returns a list
- **sum()** - applicable to only numeric tuples, returns summation of all the elements int he tuple

In [244…
```
tup = (1, 5, 3, 7, 2, 4)
```

In [233…
```
len(tup)
```

Out[233… 6

In [234…
```
min(tup)
```

Out[234… 1

In [235…
```
max(tup)
```

Out[235… 7

In [236…
```
sorted(tup)
```

Out[236… [1, 2, 3, 4, 5, 7]

In [237…
```
sum(tup)
```

Out[237… 22

Ex. WAP to reverse the tuple

In [246…
```
tup = (1, 4, 3, 2, 6, 5, 8)
tup[::-1]
```

Out[246… (8, 5, 6, 2, 3, 4, 1)

Sorting - ASC

In [247…
```
tuple(sorted(tup))
```

Out[247… (1, 2, 3, 4, 5, 6, 8)

Sorting - DESC

In [248…
```
tuple(sorted(tup, reverse=True))
```

Out[248… (8, 6, 5, 4, 3, 2, 1)

Tuple Methods

- **tup.index(`object`)** - returns the index position of first occurence of the `object`

- **tup.count(`object`)** - returns the `count` of number of times the object is repeated in the tuple

In [239…
```
tup = ('car', 'bike', 'house',  'car', 'aeroplane', 'train', 'car')
```

In [240…
```
tup.index('car')
```

Out[240… 0

In [241…
```
tup.count('car')
```

Out[241… 3

Unpacking Tuples

In [249…
```
tup = 1, 2, 3  # packing of tuples
tup
```

Out[249… (1, 2, 3)

In [250…
```
a, b, c = tup
a
```

Out[250… 1

In [251…
```
name, age = "Jane", 30  # unpacking tuples
name
```

Out[251… 'Jane'

Examples

Ex. WAP to print the tuple and the sum of subtuples

In [252…
```
tup = ((3,9),(1,2,4),(1,4,8),(2,3))
for i in tup :
    print(i, " - ", sum(i))
```

```
(3, 9)  -  12
(1, 2, 4)  -  7
(1, 4, 8)  -  13
(2, 3)  -  5
```

Ex. WAP to print the word in the tuple and its len.

In [ ]:
```
tup = ('car', 'bike', 'house', 'aeroplane')
```

Ex. WAP to print sum of all the numbers in above tuple.

In [ ]:
```

```
