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 – 3.11.x
- Download it from here - https://www.python.org/download/releases/3.0/

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 [ ]:
```
10 # int
```

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

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

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

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

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

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

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

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

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

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

In [4]:
```
{1 : "Jane", 2:"George", 3:"Sam"} # dictionary
```

Out[4]:
```
{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 [7]:
```
name = "Jane"
```

In [10]:
```
print("Welcome", name)
```

```
Welcome 10
```

In [9]:
```
name = 10
```

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

In [11]:
```
a = True
print(type(a))
```

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

In [12]:
```
a = 10
print(type(a))
```

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

In [13]:
```
a = 10.5
print(type(a))
```

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

In [14]:
```
a = "abcd"
print(type(a))
```

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

In [15]:
```
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 [16]:
```
name = input("Enter your name - ")
print("Welcome", name)
```

```
Welcome Jack
```

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

In [47]:
```
num1 = int(input("Enter a number - "))
num2 = int(input("Enter a number - "))
num1 + num2
```

Out[47]: 9

In [20]:
```
type(num1)
```

Out[20]:
```
str
```

Note - input() will always store the value in str format

Data Type Conversion

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

In [21]:
```
a = 10  # int
b = 2.5 # float
a + b
```

Out[21]:
```
12.5
```

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

In [65]:
```
a = "10"  # str
b = 5     # int

int(a) + b
```

Out[65]:
```
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 [23]:
```
x = 10.8 # float
int(x)
```

Out[23]:
```
10
```

In [24]:
```
x = "10" # int value in str format
int(x)
```

Out[24]: 10

In [25]:
```
x = True # bool
int(x)
```

Out[25]: 1

In [26]:
```
x = False # bool
int(x)
```

Out[26]: 0

In [27]:
```
x = "abcd" # str
int(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[27], line 2
      1 x = "abcd" # str
----> 2 int(x)

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

In [28]:
```
x = "10.8" # float value in str format
int(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[28], 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 [29]:
```
x = True # bool
float(x)
```

Out[29]: 1.0

In [30]:
```
x = False # bool
float(x)
```

Out[30]: 0.0

In [31]:
```
x = 10 # int
float(x)
```

Out[31]: 10.0

In [32]:
```
x = "10.8" # float value in str format
float(x)
```

Out[32]: 10.8

In [33]:
```
x = "abcd" # str
float(x)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[33], line 2
      1 x = "abcd" # str
----> 2 float(x)

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

`str()` conversion

In [35]:
```
x = 10
str(x)
```

Out[35]: '10'

In [36]:
```
lst = [1,2,3,4,5]
str(lst)
```

Out[36]: '[1, 2, 3, 4, 5]'

In [37]:
```
print(lst)
```

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

`bool()` conversion

In [38]:
```
x = 0
bool(x)
```

Out[38]: False

In [39]:
```
x = 1
bool(x)
```

Out[39]: True

In [40]:
```
x = 0.0
bool(x)
```

Out[40]: False

In [41]:
```
x = 1.0
bool(x)
```

Out[41]: True

In [42]:
```
x = 10
bool(x)
```

Out[42]: True

In [43]:
```
x = "abc"
bool(x)
```

Out[43]: True

In [44]:
```
x = "True"
bool(x)
```

Out[44]: True

In [45]:
```
x = "False"
bool(x)
```

Out[45]: True

In [46]:
```
x = None
bool(x)
```

Out[46]: False

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

print() function

- used to display output on the console
- while using jupyter notebook - last line of the code is by default printed/displayed as output (raw output)
- to print a line within the code use print()
- while writing code using .py file, print() is mandatory for every print statement

In [53]:
```
var = "Jane"
var
```

Out[53]: 'Jane'

In [50]:
```
print(var)
```

```
Jane
```

In [69]:
```
a = 10
b = 20
c = a + b
print("Addition of", a, "and", b, "=", c)
```

```
Addition of 10 and 20 = 30
```

In [55]:
```
print("Addition of", a, "and", b, "=", c, sep = "_") # sep - decides the delimiter
```

```
Addition of_10_and_20_=_30
```

In [56]:
```
print("Abcd")
print("PQR")
```

```
Abcd
PQR
```

In [58]:
```
print("Abcd", end = ", ") # end - decides which character to insert at the end of p
print("PQR")
print("XYZ")
```

```
Abcd, PQR
XYZ
```

formatted string

In [61]:
```
sales = 900
tax = input(f"Enter tax for sales price - {sales}")
```

In [74]:
```
f"Addition of {a} and {b} = {int("10")+int(b)}"
```

Out[74]:
```
'Addition of 10 and 20 = 30'
```

In [71]:
```
print(f"Addition of {a} and {b} = {a+b}")
```

```
Addition of 10 and 20 = 30
```

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 [75]:
```
a = 10
b = 6
```

In [76]:
```
print("Addition - ", a+b)
```

```
Addition -  16
```

In [77]:
```
print("Substraction - ", a-b)
```

```
Substraction -  4
```

In [78]:
```
print("Multiplication - ", a*b)
```

```
Multiplication -  60
```

In [79]:
```
print("Division - ", a/b) # returns result in float format
```

```
Division -  1.6666666666666667
```

In [80]:
```
print("Floor Division - ", a//b) # returns integer part of the division
```

```
Floor Division -  1
```

In [81]:
```
print("Exponential - ", a**b) # returns the result of a to the power b
```

```
Exponential -  1000000
```

In [82]:
```
print("Modulous - ", a%b) # returns remainder of the division
```

```
Modulous -  4
```

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 [83]:
```
a = 10
b = 7
```

In [84]:
```
print(a < b)
```

```
False
```

In [85]:
```
print(a <= b)
```

```
False
```

In [86]:
```
print(a > b)
```

```
True
```

In [87]:
```
print(a >= b)
```

```
True
```

In [88]:
```
print(a==b)
```

```
False
```

In [89]:
```
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 [92]:
```
num = int(input("Enter a number - "))
num > 10
```

Out[92]:
```
False
```

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 [93]:
```
num = int(input("Enter a number - "))
num % 5 == 0
```

Out[93]:
```
True
```

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 [94]:
```
num = int(input("Enter a number - "))
num > 10 and num % 5 == 0
```

Out[94]:
```
True
```

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

In [95]:
```
num = int(input("Enter a number - "))
num > 10 or num % 5 == 0
```

Out[95]:
```
True
```

In [96]:
```
not True
```

Out[96]:
```
False
```

In [97]:
```
not False
```

Out[97]:
```
True
```

In [98]:
```
not (10 - 5 * 2)  # example of orperator precedence and implicit conversion
```

Out[98]:
```
True
```

In [102…
```
x = int(input())
bool(x)
```

Out[102…
```
False
```

Examples of implicit conversions

In [107…
```
num = 17
if num % 2 == 0 :
    print("even")
else:
    print("odd")
```

```
even
```

In [109…
```
num = 16
if num % 4 :
    print("Not divisible")
else:
    print("Divisible")
```

```
Divisible
```

In [110…
```
if 10 :
    print("yes")
```

```
yes
```

In [ ]:
```
var = True
# some code to modify var

if var:

```

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 [111…
```
var = 10
```

In [112…
```
var += 1  # - var = var + 1
```

Ex. WAP to calculate BMI of a person.

In [113…
```
weight = float(input("Enter weight in kgs - "))
height = float(input("Enter 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 [114…
```
hrs = int(input("Number of hours - "))
rate = int(input("Rate per hr - "))
gross_pay = hrs * rate
print(gross_pay)
```

```
40000
```

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

In [115…
```
import math
math.sqrt(64)
```

Out[115… 8.0

In [116…
```
import math as m
m.sqrt(64)
```

Out[116… 8.0

In [121…
```
from math import sqrt  # avoid this way of import, results in ambiguity
sqrt(64)
```

Out[121… 8.0

In [122…
```
sqrt = 10
```

In [123…
```
sqrt * 5
```

Out[123… 50

In [124…
```
sqrt(64)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[124], line 1
----> 1 sqrt(64)

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

In [1]:
```
print("hello")
```

```
hello
```

In [4]:
```
import math as m
ht = 4
sqrt = 3
hypt = m.sqrt(ht**2 + sqrt**2)
hypt
```

Out[4]: 5.0

In [3]:
```
(ht**2 + base**2) ** 0.5
```

Out[3]: 5.0

To view functions in a module -

In [6]:
```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'a
sin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysign', 'cos',
'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'fact
orial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'lo
g1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'r
emainder', 'sin', 'sinh', 'sqrt', 'sumprod', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

In [9]:
```
math.pi
```

Out[9]: 3.141592653589793

In [ ]:

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 [10]: "A" in "ABCD"
```

```
Out[10]: True
```

```
In [11]: "A" not in "ABCD"
```

```
Out[11]: False
```

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

```
In [14]: name = input("Enter name - ")
         names = ["Jane", "George", "Sam"]
         name in names
```

```
Out[14]: False
```

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

```
In [16]: ch = input("Enter a character - ")
         ch in "aeiou"
```

```
Out[16]: True
```

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 check if entered number is positive or negative

In [18]:
```
num = int(input("Enter a number - "))
if num > 0 :
    print("Positive")
else:
    print("Negative")
```

```
Negative
```

Ex. WAP to check if a number is positive, negative or zero.

In [21]:
```
num = int(input("Enter a number - "))
if num > 0 :
    print("Positive")
elif num < 0:
    print("Negative")
else:
    print("zero")
```

```
Negative
```

Ex. WAP to check if user has entered a integer, if yes then check if it is positve or negative number.

In [25]:
```
num = input("Enter a number - ")
if num.isdigit() :
    num = int(num)
    if num > 0 :
        print("Positive")
    elif num < 0:
        print("Negative")
    else:
        print("zero")
else:
    print("Invalid value")
```

```
Positive
```

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

In [29]:
```
ch = input("Enter a character - ")
result = "Vowel" if  ch in "aeiou" else "Consonant"
result
```

Out[29]:
```
'Vowel'
```

Example - python is dynamically typed

In [32]:
```
ch = input("Enter a character - ")
result = "Vowel" if  ch in "aeiou" else 0
result
```

Out[32]:
```
0
```

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 [ ]:

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.

1. Take guess as input from user as heads or tails
2. check if it is a valid | if no print invalid
3. generate a random outcome from heads or tails (take static value in the begining)
4. compare outcome and guess
5. if equal player wins
6. else loose

In [57]:
```
import random as r
guess = input("Enter your guess as heads or tails - ").lower()
options = ("heads", "tails")
if guess in options :
    outcome = r.choice(options)
    print("Outcome - ", outcome)
    if guess == outcome :
        print("Win")
    else:
        print("Lost")
else:
    print("Invalid")
```

```
Outcome -  heads
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 [37]:
```
import random as r # as is an operator - used to assign a variable to a moudule whi
```

In [38]:
```
r.random() # generates a random value between 0 and 1
```

Out[38]:
```
0.10827813904186645
```

In [39]:
```
r.randint(1,5) # generates a random integer between the mentioned start and end val
```

Out[39]:
```
4
```

In [44]:
```
r.choice([1,2,3,4]) # generates a random value from the mentioned iterable or seque
```

Out[44]:
```
4
```

In [48]:
```
r.choice(["abc", "pqr", "xyz"])
```

Out[48]:
```
'pqr'
```

In [55]:
```
r.choice("abcde")
```

Out[55]:
```
'b'
```

In [43]:
```
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 [58]:
```
amount = 1000
outcome = r.randint(1, 14)
print("Your Score - ", outcome)
```

```
if outcome < 7 :
    amount -= (outcome * 100)
elif outcome > 7 :
    amount += (outcome * 100)
else:
    print("Congratulations!! you have hit the jackpot!!")
    amount += 1000000
print("Final Balance - ", amount)
```

```
Your Score -  4
Final Balance -  600
```

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 [108…

```
amount = 1000
choice = "yes"

while choice == "yes" :
    outcome = r.randint(1, 14)
    print("Your Score - ", outcome)

    if outcome < 7 :
        amount -= (outcome * 100)
```

```
    elif outcome > 7 :
        amount += (outcome * 100)
    else:
        print("Congratulations!! you have hit the jackpot!!")
        amount += 1000000
        break

    if amount <= 600 :
        choice = input("Insufficient funds. Do you wish to top-up? yes/no - ").lowe
        if choice == "yes" :
            amount += 1000
            continue
        break

    print(f"Your current balance is {amount}.")
    choice = input("Do you wish to play again? yes/no - ").lower()

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

```
Your Score -  5
Final Balance -  500
```

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 [72]:
```
numbers = [1, 2, 3, 4, 5]
for i in numbers :
```

```
    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 [73]:
```
numbers = [1, 2, 3, 4, 5]
for i in numbers :
    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 [78]:
```
word = input("Enter a word - ")
for ch in "aeiou" :
    if ch in word.lower() :
        print(ch)
```

```
a
i
o
```

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 [80]:
```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11, 43]
product = 1
total = 0

for i in numbers :
    product *= i
    total += i
print(product, total)
```

```
54489600 91
```

In [81]:
```
sum(numbers)
```

Out[81]: 91

In [83]:
```
import math as m
m.prod(numbers)
```

Out[83]: 54489600

Ex. WAP to perform product of first 10 natural numbers

In [93]:
```
m.prod(range(1, 11))
```

Out[93]: 3628800

range([start], stop, [step])

- returns a **sequence object** of intergers from start to stop -1
- `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 [84]:
```
range(1, 11)  # 1 - 10
```

Out[84]:
```
range(1, 11)
```

In [86]:
```
for i in range(1,11):
    print(i, end = ", ")
```

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

In [87]:
```
for i in range(11):
    print(i, end = ", ")
```

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

In [88]:
```
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 [89]:
```
for i in range(1, 50, 5):
    print(i, end = ", ")
```

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

In [90]:
```
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 [91]:
```
for i in range(50, -1, -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, 0,
```

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 [94]:
```
num = int(input("Enter a number - "))
for i in range(1, 11) :
    print(f"{num} x {i} = {num*i}")
```

```
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

Ex. WAP to print prime number between 1 - 100

In [97]:
```
num = 170
flag = True # assuming num is prime
for i in range(2, num) :
    if num % i == 0 :
        flag = False
        break
if flag :
    print("prime")
else:
    print("not prime")
```

```
not prime
```

for .. else

- if a for loop is `terminated` abruptly using a `break` statement then else block is `NOT` executed
- if for loop is executed successfully till its last iteration, then else block is `executed`

In [101…
```
for i in range(1, 10) :
    print(i)
    if i == 7 :
        print("Terminated")
        break
else :
    print("Else block executed")
print("Outside for loop")
```

```
1
2
3
4
5
6
7
Terminated
Outside for loop
```

In [100…
```python
for i in range(1, 6) :
    print(i)
    if i == 7 :
        print("Terminated")
        break
else :
    print("Else block executed")

print("Outside for loop")
```

```
1
2
3
4
5
Else block executed
Outside for loop
```

In [103…
```python
num = 17
for i in range(2, num) :
    if num % i == 0 :
        print("not prime")
        break
else :
    print("prime")

print("outside the for loop")
```

```
prime
outside the for loop
```

In [104…
```python
num = 170
for i in range(2, num) :
    if num % i == 0 :
        print("not prime")
        break
else :
    print("prime")

print("outside the for loop")
```

```
not prime
outside the for loop
```

In [106…
```python
for num in range(2, 101) :
    for i in range(2, num) :
        if num % i == 0 :
```

```
            break
    else :
        print(num, end = " ")
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```
