Introduction to Python

Python Features

- Python is an easy to learn, high-level programming language.
- It is known for its simplicity, readability, and versatility
- <u>Features of Python is</u>
  - Simple and Easy to Learn
  - Interpreted Language
  - Dynamically Typed
  - High-Level Language
  - Supports Object-Oriented Programming
  - Extensive Standard Library
  - Interactive Programming language
  - Its Free, powerful & portable
  - Used in Data Science, AI, Web Development, and More

Simple and Easy to Learn

- Python syntax is clear, concise, and readable.
- Structure and syntax are pretty intuitive and easy to grasp.
- Example of a simple Python program:
  - print('Hello, World!')
- This is all you need to display 'Hello, World!' in Python.

Interpreted Language

- Python is an interpreted language, meaning code is executed line by line.
- There is no need for compilation before execution.
- Example of Python's interactive nature:
  - >>> print('Python is interpreted!')
  - Output: Python is interpreted!
- This is executed in the Python shell.

Dynamically Typed

- Python does not require explicit type declarations for variables.
- The type is determined at runtime based on the assigned value.
- Example:
  - x = 10 # Integer
  - x = 'Hello' # Now a String
  - print(x)
- The same variable `x` can hold different types during execution.

Object-Oriented

- Python supports object-oriented programming (OOP) concepts.
- This includes classes, objects, inheritance, polymorphism, etc.
- Example of a simple Python class:
  - class Animal:
  - def __init__(self, name):
  - self.name = name
  - def speak(self):
  - return 'I am ' + self.name
  - dog = Animal('Dog')
  - print(dog.speak())
- Output: I am Dog

Extensively Used in Data Science and AI

- Python is a top choice for data science and artificial intelligence (AI) applications.
- Libraries like NumPy, Pandas, and TensorFlow make Python versatile in these fields.
- Example: Using Pandas to read a CSV file:

```
import pandas as pd
data = pd.read_csv('data.csv')
print(data.head())
```

- This code reads a CSV file and displays the first few rows.

Free , Powerful & portable

- Python is open source and downloading python and installing python is free and easy.

- Dynamic typing
- Built-in types and tools
- Library utilities
- Third party utilities (e.g. Numeric, NumPy, sciPy)
- Automatic memory management

- Python runs virtually every major platform used today
- Python programs will run in exactly the same manner, irrespective of platform.

Python in Web Development

- Python is widely used for web development with frameworks like Django and Flask.
- Case Study: A company builds a content management system (CMS) using Django.
- With Python, they can build scalable and secure web applications quickly.
- Example Django Code for a Simple View:

```
from django.http import HttpResponse
def hello(request):
return HttpResponse('Hello, Django!')
```

- This simple view sends a 'Hello, Django!' message as a response.

Indentation

- Python uses indentation to define code blocks
- Indentation is mandatory and critical for code structure
- Generally 4 spaces per level of indentation
- Ex

```
if(x%2==0):
    print('Even no')
```

Strongly Typed Features

- Python is a strongly typed language
- **Strongly Typed** - variables do have a type and that the type matters when performing operations on a variable
- Implicit type conversion is not allowed
- Variables need explicit type conversion
- Example:

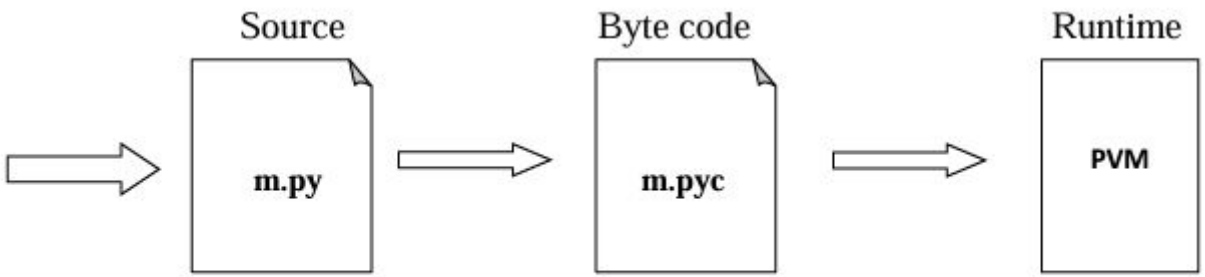
```
x = 10;
y = '5';
result = x + int(y)
```

Python Shell

- The Python shell (REPL) allows interactive testing
- It evaluates expressions and displays results immediately

Python Code Execution

- Source code you type is translated to byte code, which is then run by the Python Virtual Machine (PVM).
- Code is automatically compiled, but then it is interpreted.



Source code extension is .py
Byte code extension is .pyc (Compiled python code)

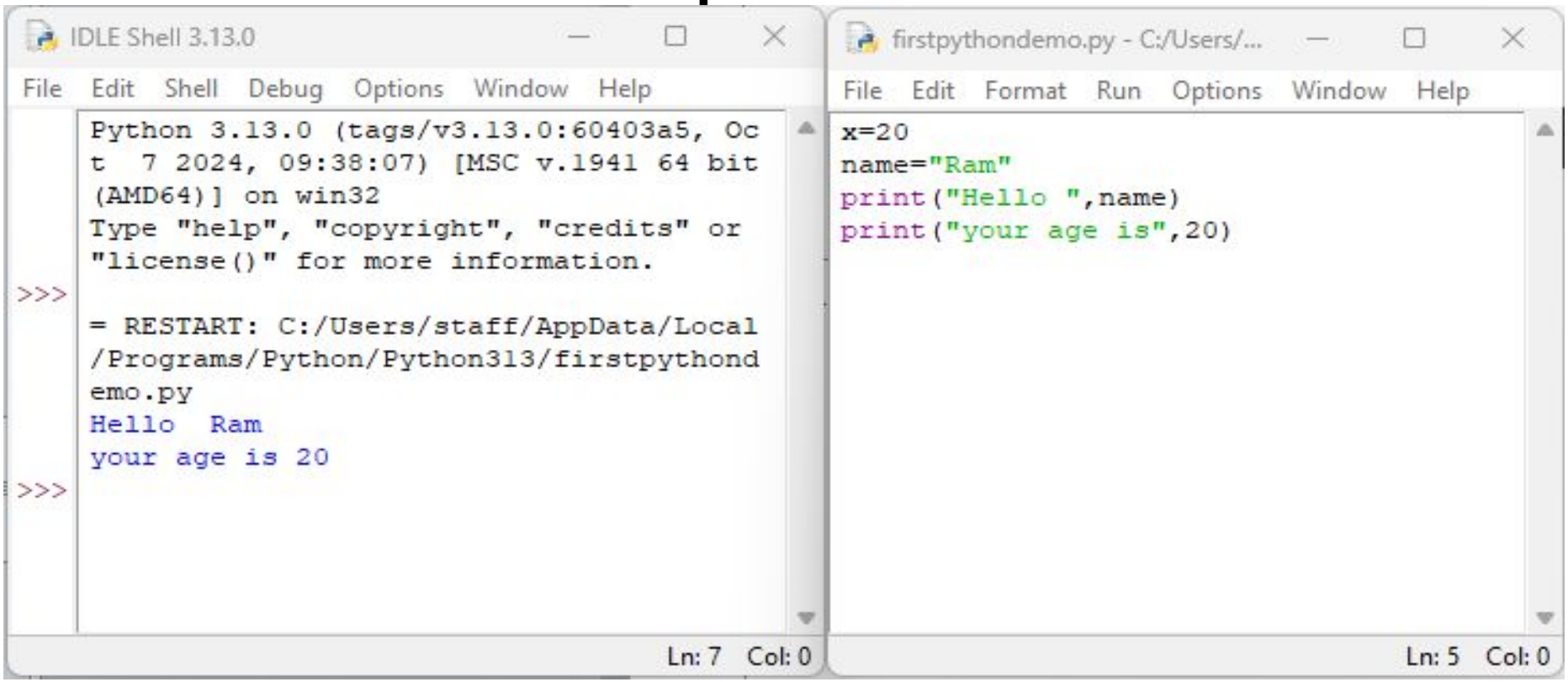
- There are two modes for using the Python interpreter:
  - Interactive Mode
  - Script Mode

Interactive Mode

Command Prompt - py

```
C:\Users\staff>py
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello Friends")
Hello Friends
>>> x=5
>>> print(x)
5
>>>
```

Script Mode



- Create new file with the .py extension, and use the interpreter to execute the contents of the file.
- To execute the script press Run menu from file

Structure of a Python Program

A Python program typically starts with import statements
It can contain function definitions, classes, and statements
Programs end with a return or exit statement
Python program include following thing

- Import Statements
- Functions and Methods
- Variables and Data Assignments
- Control Flow Statements (if-else)
- Loops (for, while)
- Input/Output Operations
- With these components, you can create dynamic and functional Python programs.

Elements of Python

- Variables
- Constants
- Operators
- Expressions
- Functions

Basic Data Types

- int (Integer)
- float (Floating-point number)
- str (String)
- bool (Boolean)
- Complex

Int data type

- Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

integer1.py - C:/Users/staff/AppData/Loca...

File Edit Format Run Options Window Help

```
#integer datatype demo
x=5
print (x)
print(type(x))
x=9988776655
print (x)
print(-6,"   ",type(-6))
print(0,"    ",type(0))
print(0b101,"    ",type(0b101))
print(0B11,"    ",type(0B11))   #binary
print(0x2f,"    ",type(0x2f))   #hexadecimal
print(0o27,"    ",type(0o27))   #octal
print(0o29,"    ",type(0o29))
```

SyntaxError

invalid digit '9' in octal literal

OK

IDLE Shell 3.13.0

File Edit Shell Debug Options

```
Python 3.13.0 (tags/v3
AMD64)] on win32
Type "help", "copyright
>>>
== RESTART: C:/Users/s
5
<class 'int'>
9988776655
-6     <class 'int'>
0      <class 'int'>
5      <class 'int'>
3      <class 'int'>
47      <class 'int'>
23      <class 'int'>
>>>
```

Float Data type

- Floating point number include positive or negative & containing one or more decimals.
- Float can also be scientific numbers with an "e" to indicate the power of 10.

float1.py - C:/Users/staff/AppData/Loc...

File Edit Format Run Options Window Help

```
#Float datatype demo
x=5.0
print (x)
print(type(x))
x=11111111111111.22222222222
print(x,"  ",type(x))
print(1.1," ",type(1.1))
print(2/5," ",type(2/5))
print(1.," ",type(1.))
print(.2," ",type(.2))
print(1e2," ",type(1e2))
print(1E3," ",type(1E3))
print(1.1e2," ",type(1.1e2))
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window Help

```
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:3
8:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for
more information.
>>>
=== RESTART: C:/Users/staff/AppData/Local/Programs/Pyt
hon/Python313/float1.py ==
5.0
<class 'float'>
11111111111111.223    <class 'float'>
1.1   <class 'float'>
0.4   <class 'float'>
1.0   <class 'float'>
0.2   <class 'float'>
100.0   <class 'float'>
1000.0   <class 'float'>
110.0   <class 'float'>
>>>
```

Str Data type

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- 'hello' is the same as "hello".
- para=" " " hello

  Hello

  Hello

  Hello" " ".

Here " para " is a multiline string variable

string1.py - C:/Users/staff/AppData/Local/Programs/Python/Python313/string1.py (3.13.0)

File Edit Format Run Options Window Help

```python
#string Data tpe

print("Hello", type("Hello"))
print('Hello', type('Hello'))

#include special character ' " widhout using escape character
print("It's fantastic")
print('"Arise, awake, and stop not till the goal is reached"-Swami Vivekanand')
#With using escape character \
print('It\'s fantastic')
print("\"Arise, awake, and stop not till the goal is reached\"-Swami Vivekanand")
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window Help

```
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:/Users/staff/AppData/Local/Programs/Python/Python313/string1.py ==
Hello <class 'str'>
Hello <class 'str'>
It's fantastic
"Arise, awake, and stop not till the goal is reached"-Swami Vivekanand
It's fantastic
"Arise, awake, and stop not till the goal is reached"-Swami Vivekanand
>>>
```

Bool data type

- Objects of Boolean type may have one of two values, True or False:
- example

```
>>> type(True) <class 'bool'>
>>> type(False) <class 'bool'>
```

Complex Data type

- It contain real and imaginary part like a+bj where a,b are two constant and j is imaginary part

complex1.py - C:/Users/staff/AppData/Local/Programs/P

File Edit Format Run Options Window Help

```
#complex number
print(2j,"  ", type(2j))
print(5+2j," ", type(5+2j))
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window Help

```
Python 3.13.0 (tags/v3.13.0:60403a5,
AMD64)] on win32
Type "help", "copyright", "credits" c
>>>
== RESTART: C:/Users/staff/AppData/Lc
2j    <class 'complex'>
(5+2j)   <class 'complex'>
>>>
```

Variables

- Variables are nothing but **reserved memory locations** to store values This means that when you create a variable you reserve some space in memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory
- They are dynamically typed in Python
- Rules for Python variables:
  - A variable name must start with a letter or the underscore character
  - A variable name cannot start with a number
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case-sensitive (age, Age and AGE are three different variables)

Assigning Values to Variables:

- Python variables do not need explicit declaration to reserve memory space.
- The declaration happens automatically when you assign a value to a variable.
- The equal sign (=) is used to assign values to variables.

variable1.py - C:/Users/staff/AppData/L

File Edit Format Run Options Win

```
#variable declaration

#single value allocation
a=5
print(a)

#multiple value allocation
a,b,c= 5,6, "abc"
print(a,b)
print("hello",c)

a=b=c=7
print(a,b,c)
```

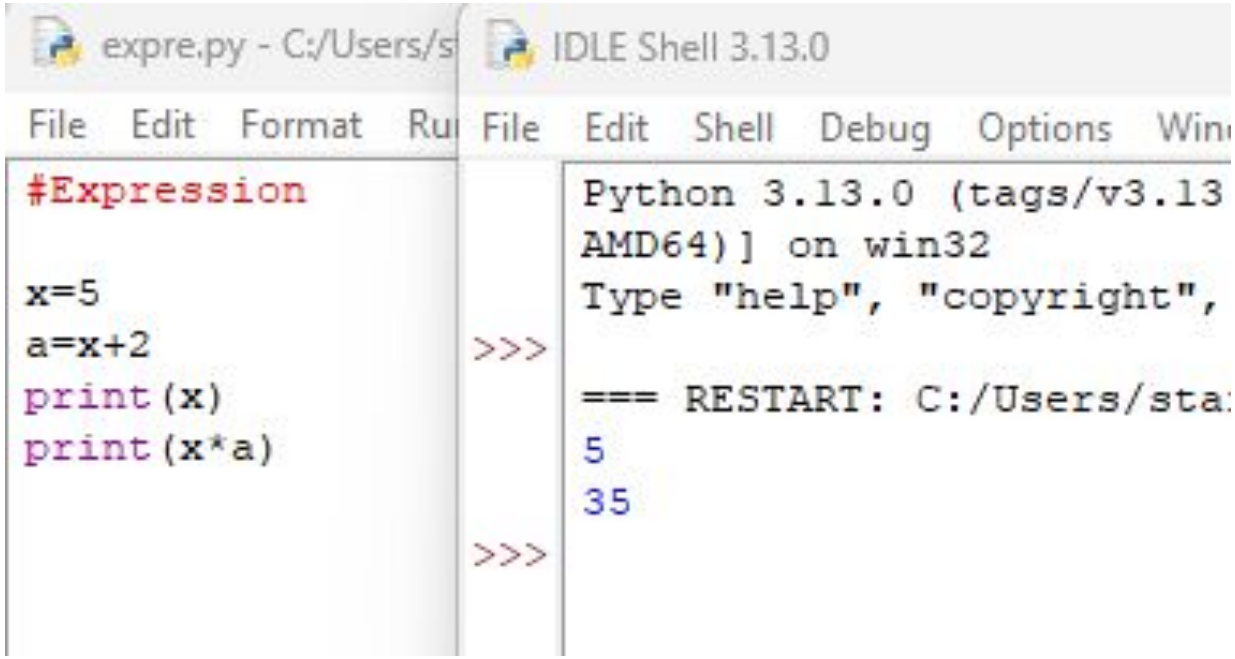
IDLE Shell 3.13.0

File Edit Shell Debug Options Window He

```
Python 3.13.0 (tags/v3.13.0:604
AMD64)] on win32
Type "help", "copyright", "cred
>>>
= RESTART: C:/Users/staff/AppDat
5
5 6
hello abc
7 7 7
>>>
```

Expressions

- An expression is a combination of values, variables, and operators
- It evaluates to produce a result



Operator

Type of operator	Operator list	Example
Arithmetic operators	+ - * / % ** //	a + b
Assignment operators	= += -= &= |= >>= := etc.	b = 5
Comparison operators	== != > < >= <=	a > b
Logical operators	And or not	a > b and a > c
Identity operators	Is, is not	a is y
Membership operators	In, not in	a in y
Bitwise operators	& | ^ ~ << >>	a & y
Increment and decrement operator	+= -=	a += 1
Ternary operator	(True) if (condition) else (false)	

Ternary operator

- **Ternary If Else**

  min = "a is minimum" if a < b else "b is minimum"

- **Ternary nested if**

  print("Both are equal" if a == b else "a is greater“ if a > b else "b is greater")

- **Ternary with print function**

  print(a,"is minimum") if (a < b) else print(b,"is minimum")

Control Statement

- If
- If else
- If elif else
- Match case

If statement

- The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.
- Syntax:
  - if condition(boolean expression):

    Statements
- If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed.
- Here indentation required to specify the multiple statements inside the true part.

if1.py - C:/Users/staff/AppData/Local/Programs/Python/Python313/if

File Edit Format Run Options Window Help

```python
x=int(input('enter any number'))
if(x>0):
    print(x ,'is positive number')
print('Normal statement')
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window

```
Python 3.13.0 (tags/
AMD64)] on win32
Type "help", "copyri
>>>
==== RESTART: C:/Use
enter any number5
5 is positive number
Normal statement
>>>
==== RESTART: C:/Use
enter any number-2
Normal statement
```

If else

- if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.
- Syntax:

```
if condition(boolean expression):
        Statements 1
else
        statements 2
```

- If the boolean expression evaluates to TRUE, then statement1 execute or in case of false statement2 execute.

if2.py - C:/Users/staff/AppData/Local/Programs/Python/Python313/if

File Edit Format Run Options Window Help

```
x=int(input('enter any number'))
if(x>0):
    print(x ,'is positive number')
else:
    print(x, 'is negative number')
print('Normal statement')
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window H

```
Python 3.13.0 (tags/v3
AMD64)] on win32
Type "help", "copyrigh
>>>
==== RESTART: C:/Users
enter any number5
5 is positive number
Normal statement
>>>
==== RESTART: C:/Users
enter any number-2
-2 is negative number
Normal statement
```

if elif else

- if, elif, and else are used for conditional branching. These statements allow you to execute different blocks of code depending on the conditions that evaluate to True or False. Here's how each one works:
- Syntax:

```
if condition1:
    # Code to execute if condition1 is True
elif condition2:
    # Code to execute if condition1 is False, but condition2 is True
else:
    # Code to execute if neither condition1 nor condition2 are True
```

if3.py - C:/Users/staff/AppData/Local/Programs/Python/Python31

File Edit Format Run Options Window Help

```
x=int(input('enter any number'))
if(x>0):
    print(x ,'is positive number')
elif(x<0):
    print(x, 'is negative number')
else:
    print(x, 'is zero')
print('Normal statement')
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window

```
Python 3.13.0 (tags/v
AMD64)] on win32
Type "help", "copyrig
>>>
==== RESTART: C:/Use
enter any number4
4 is positive number
Normal statement
>>>
==== RESTART: C:/Use
enter any number0
0 is zero
Normal statement
```

Match case

- match statement allows you to perform **structural pattern matching**, which is a more powerful and readable alternative to using multiple if/elif/else conditions.
- The match statement is particularly useful when you need to match patterns in data structures like lists, tuples, dictionaries, or even simple variables.
- Syntax:

```
match variable:
    case pattern1:
        # Code to execute if variable matches pattern1
    case pattern2:
        # Code to execute if variable matches pattern2
    case _:
        # Code to execute if no patterns match (default case)
```

match1.py - C:\Users\staff\AppData\Local\Programs\Python\Python31

File Edit Format Run Options Window Help

```
x = int(input("Enter a number: "))
match x:
    case x if x > 0:
        print("Positive")

    case x if x < 0:
        print("Negative")
    case _:
        print("Zero")
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Wind

```
Python 3.13.0 (tag
AMD64)] on win32
Type "help", "copy
>>>
=== RESTART: C:\Us
Enter a number: 5
Positive
>>>
```

Loop Controls: while

- A while loop repeatedly executes a block of code as long as the condition is True.
- If the condition is False at the start, the body of the loop is not executed.
- Common loop controls include break (to exit the loop) and continue (to skip the current iteration).
- ***Syntax:*** ***while*** *expression:*
  statement(s)

while1.py - C:/Users/staff/

File Edit Format Run C

```
i=0;
while(i<5):
    i=i+1;
    print(i)
print('bye')
```

```
>>>
1
2
3
4
5
bye
```

Loop Controls: for

- The for loop is used to iterate over elements of a sequence (list, string, tuple, range, etc.).
- It's an efficient way to perform repetitive tasks over a known sequence of items.
- You can control the flow of the loop with statements like break, continue.
- **Syntax:** for item in sequence:

  Statements

for1.py - C:/Users/staff/AppData/Local/Prog

File Edit Format Run Options Window

```
x="Hello"
for i in x:
    print(i," letter")
```

```
>>>
H  letter
e  letter
l  letter
l  letter
o  letter
```

Range function

- range() function is used to generate a sequence of numbers.
- The range() function does not generate all the values at once; instead, it returns a **range object**, which is an iterator that generates values on-demand, making it memory efficient for large sequences.
- The range() function can take up to three arguments:
- **Syntax:**

  range(start, stop, step)

  - **start** (optional): The value where the sequence starts. The default is 0 if not specified.
  - **stop**: The value where the sequence stops (but does not include). This is a required argument.
  - **step** (optional): The difference between each pair of consecutive numbers in the sequence. The default is 1 if not specified. Negative step generate number sequence in reverse.

range1.py - C:\Users\staff\AppData\Local\Programs\Python\Python313\range1.py (3.13.0)

File Edit Format Run Options Window Help

```
#print 0 to 6
x=range(6)
for i in x:
    print(i, end=' ')
print('\n====================')
#print 5 to 10
x=range(5,11)
for i in x:
    print(i, end=' ')
print('\n====================')
#print 1, 3, 5, 7 ,9
x = range(1, 10, 2)
for n in x:
  print(n, end=' ')
print('\n====================')
#accessing range value with perticular index
print('-1',x[-1])
print('0',x[0])
print('1',x[1])
print('3',x[3])
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window

```
Python 3.13.0 (tags/
AMD64)] on win32
Type "help", "copyri
>>>
=== RESTART: C:\User
0 1 2 3 4 5
====================
5 6 7 8 9 10
====================
1 3 5 7 9
====================
-1 9
0 1
1 3
3 7
>>>
```

Break statement

- break statement is used to **exit** or **terminate** a loop prematurely either in while loop or for loop
- It is useful when you want to stop further iterations when a specific condition is met..
- It is commonly used in situations such as searching for an item in a collection, breaking out of infinite loops, or handling early termination in loops

breakif.py - C:\Users\staff\AppData\Local\Programs\Python\Py

File Edit Format Run Options Window Help

```
x=int(input('enter any number'))
i=0;
while(i<x):
    i=i+1;
    if(i==3):
        break
    print(i)
print('bye')
```

```
>>>
enter any number5
1
2
bye
```

continue statement

- **continue** is used to skip the current iteration in a loop and continue with the next iteration.
- It helps control the flow of the loop when certain conditions are met, allowing you to avoid unnecessary processing within that iteration.
- It can be used in both for and while loops.

breakif.py - C:\Users\staff\AppData\Local\Programs\Python\Py

File Edit Format Run Options Window Help

```
x=int(input('enter any number'))
i=0;
while(i<x):
    i=i+1;
    if(i==3):
        continue
    print(i)
print('bye')
```

```
>>>
enter any number5
1
2
4
5
bye
```

pass statement

- pass statement is a **null operation** — it does nothing when executed. It is often used as a placeholder in code where a statement is syntactically required but you do not want to execute any action.
- It can use with if statement, loop statement, functions and class
- It prevent syntax error when block is required

ifpass.py - C:\Users\staff\AppData\Local\Programs\Python\Pyt

File Edit Format Run Options Window Help

```
x=int(input('enter any no'))
if(x>0):
    pass
else:
    print("false")
print('bye')
```

```
>>>
=== RESTART: C:
enter any no5
bye
>>>
=== RESTART: C:
enter any no-3
false
bye
```