Programming in Python

Module 1

Rijin IK

Assistant Professor
Department of Computer Science and Engineering
Vimal Jyothi Engineering College
Chemperi

February 4, 2024

Outline

Introduction to Python

Python

- Python is a high-level, interpreted, and general-purpose programming language.
- It was created by Guido van Rossum and first released in 1991.
- Python has become one of the most popular programming languages in the world due to its simplicity, readability, and versatility.

Introduction to Python

Some of the features which make Python so popular are as follows:

- It is a general purpose programming language which can be used for both scientific and non scientific programming.
- It is a platform independent programming language.
- It is a very simple high level language with vast library of add-on modules.
- It is excellent for beginners as the language is interpreted, hence gives immediate results.
- The programs written in Python are easily readable and understandable.
- It is suitable as an extension language for customizable applications.
- It is easy to learn and use.
- Dynamically typed and garbage collected
- Supports procedural, object oriented and functional programming.

Running Python Program

Downloading and installing Python

- Download the latest release of Python from **www.python.org** depending on your OS Windows,Linux or Mac
- Complete the installation.

Running Python Program

Interactive Shell

1. The interactive shell is a command-line interface where you can enter Python commands and get immediate results.
2. You can open the Python interactive shell by typing python in your terminal or command prompt.
3. Once in the shell, you can execute Python code line by line.

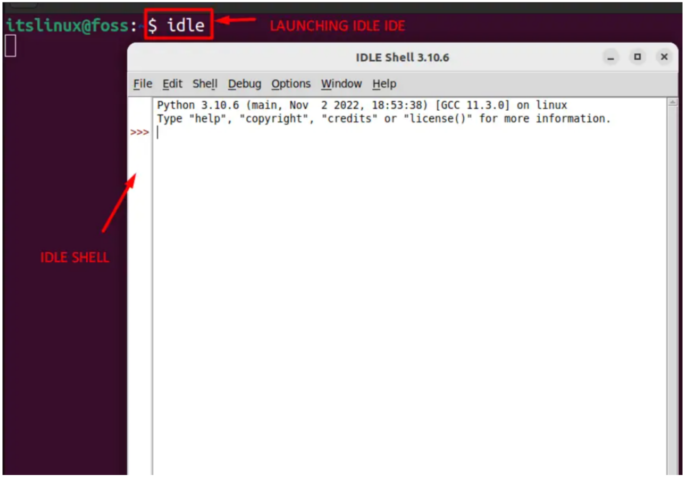
```
# Open the Python interactive shell
$ python

# Execute Python code
>>> print("Hello, World!")
Hello, World!
```

Running Python Program

IDLE (Integrated Development and Learning Environment):

- IDLE is an integrated development environment that comes with the Python installation.
- It provides a more user-friendly interface compared to the standard interactive shell.
- You can write, edit, and run Python scripts in IDLE.
- To open IDLE, you can type idle in your terminal or command prompt.
- Save your file with .py extension (Eg:test.py)

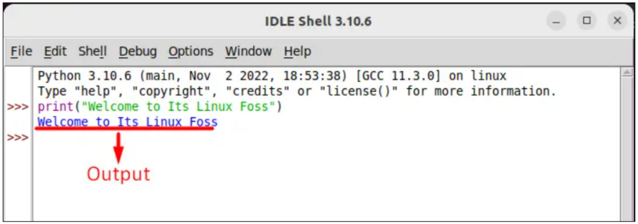
Running Python Program



```
itslinux@foss:~$ idle
```

```
Python 3.10.6 (main, Nov  2 2022, 18:53:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
```

Running Python Program



```
Python 3.10.6 (main, Nov  2 2022, 18:53:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Welcome to Its Linux Foss")
Welcome to Its Linux Foss
>>>
```

Running Python Program

You can run the script previously created(test.py) by importing it in the python command line

```
>>>import test.py
```

Running Python Program

Jupyter Notebooks:

- Jupyter Notebooks are a popular tool for interactive computing in Python.
- They allow you to create and share documents that contain live code, equations, visualizations, and narrative text.
- Jupyter Notebooks can be installed using tools like Anaconda or directly with pip.
- To open a Jupyter Notebook, run the command:

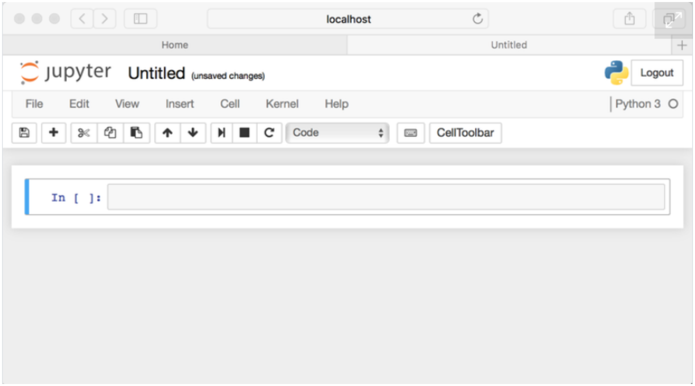
```
$ jupyter notebook
```

Running Python Program

Jupyter Notebook is great for the following use cases:

- learn and try out Python
- data processing / transon
- numeric simulation
- statistical modeling
- machine learning
- Jupyter Notebook is perfect for using Python for scientific computing and data analysis with libraries like numpy, pandas, and matplotlib.

Running Python Program



Input, Processing, and Output

Input, Processing, and Output

- Useful programs involve accepting inputs, processing them, and producing outputs.
- In terminal-based interactive programs, the keyboard serves as the input source, and the terminal display is the output destination.

Input, Processing, and Output

Print Function:

- The print function in Python is used to display output to the console or terminal
- **Syntax:**

  ```
  print(<expression>)
  ```

  - Python evaluates the expression and displays its value.
  - Strings are enclosed in quotation marks.
- **Examples:**
  1. Displaying text:
     ```
     print("Hi there")
     ```
  2. Multiple expressions:
     ```
     print(2 + 2, "hello")
     ```
- Output ends with a newline by default.
- To continue on the same line, use the end="" argument.

  ```
  # Example:
  print("This is on", end="")
  print(" the same line.")
  ```

Input, Processing, and Output

Input Function:

- Asks the user for input and waits for them to enter a value from the keyboard.
- When the user presses enter, the input function accepts the input value.
- **Syntax**

```
<variable> = input(<prompt>)
```

$< variable >$ is assigned the value entered by the user.

```
# Example:
name = input("Enter your name: ")
```

Input, Processing, and Output

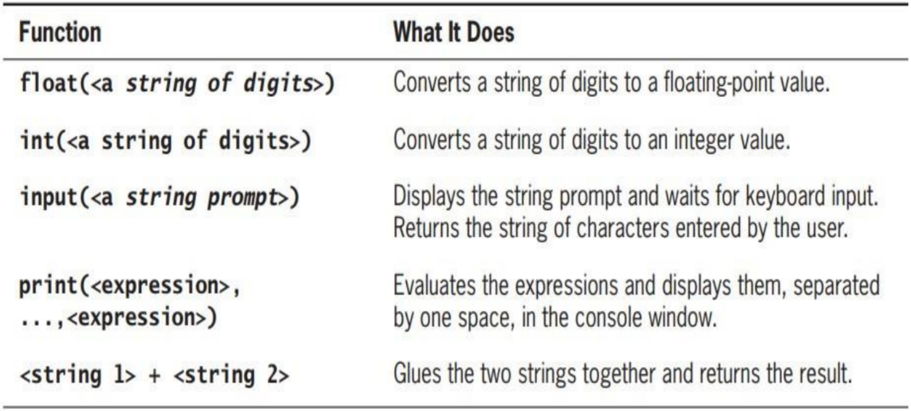
Type Conversion: Input Function and Strings:

- The input function always returns a string.
- Convert strings representing numbers to numeric types using int (for integers) or float (for floating-point numbers).

```
>>> first=int(input("enter the first number: "))
enter the first number: 12
>>> second=int(input("enter the second number: "))
enter the second number: 34
>>> print ("the sum is", first+second)
the sum is 46
```

Input, Processing, and Output

Basic Python functions for input and output



Function	What It Does
`float(<a string of digits>)`	Converts a string of digits to a floating-point value.
`int(<a string of digits>)`	Converts a string of digits to an integer value.
`input(<a string prompt>)`	Displays the string prompt and waits for keyboard input. Returns the string of characters entered by the user.
`print(<expression>, ...,<expression>)`	Evaluates the expressions and displays them, separated by one space, in the console window.
`<string 1> + <string 2>`	Glues the two strings together and returns the result.

Editing, Saving and Running a Script

Editing, Saving and Running a Script

- Trying out short Python commands is easy in the shell, but for longer and more complex programs, it's better to write, edit, and save them in files.
- Afterward, you can run these programs either in IDLE or directly from the command prompt without opening IDLE. This gives you flexibility in how you execute your Python scripts.

Editing, Saving and Running a Script

To create and run programs using this approach, follow these steps:

1. Open a new window by selecting "New Window" from the File menu in the shell.
2. In the new window, write Python expressions or statements on separate lines in the desired order.
3. Save your file by choosing File/Save.
4. It's recommended to use the .py extension, like naming it myprogram.py.

To execute your code as a Python script:

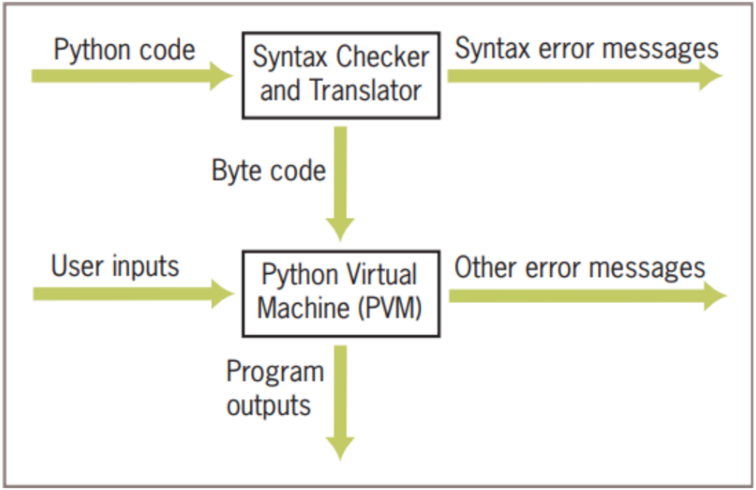
- Choose "Run Module" from the Run menu or press F5 (Windows) or Control+F5 (Mac or Linux).

How Python works

Interpreting a Python program involves

- Reading and verifying the source code,
- Translating it into bytecode,
- Sending the bytecode to the Python Virtual Machine (PVM),
- Executing the bytecode, and finally producing the desired output or handling errors if they occur.

How Python works



Detecting and Correcting Syntax Errors in Python

- The Python interpreter is adept at catching syntax errors, which occur when the code doesn't adhere to the language's rules.
- When such errors are encountered, Python stops execution and provides detailed error messages, assisting you in identifying and correcting the mistakes in your code.

Detecting and Correcting Syntax Errors in Python

Python's Error Detection:

- The Python interpreter is designed to catch errors in your code.
- Errors detected by the interpreter are known as syntax errors.

Understanding Syntax Errors:

- Syntax errors occur when your code doesn't follow the proper rules of the Python language.

Halting Execution on Syntax Errors:

- When Python comes across a syntax error in your program, it stops running and displays an error message.
- The error message indicates where the issue is located in your code.

Types of Syntax Errors:

- Python shell shows various types of syntax errors, each with its corresponding error message.
- These error messages help you identify and fix mistakes in your code.

Detecting and Correcting Syntax Errors in Python

Example:

```
y = x + 3
print(y)

Traceback (most recent call last):
File "/home/runner/new/main.py", line 1, in <module>
y = x + 3
NameError: name 'x' is not defined
```

```
result = (4 + 5 * 2
print(result)

 File "/home/runner/new/main.py", line 1
result = (4 + 5 * 2
 ^
SyntaxError: '(' was never closed\\
```

Detecting and Correcting Syntax Errors in Python

Exercises

1 Open a Python shell, enter the following expressions, and observe the results:

1. 8
2. $8 * 2$
3. $8 * *2$
4. $8/12$
5. $8//12$
6. $8/0$

Detecting and Correcting Syntax Errors in Python

Exercises

1. Write a Python program that prints (displays) your name, address, and telephone number.
2. Evaluate the following code at a shell prompt: print("Your name is", name). Then assign name an appropriate value, and evaluate the statement again.
3. Open an IDLE window, and enter the program that computes the area of a rectangle. Load the program into the shell by pressing the F5 key, and correct any errors that occur. Test the program with different inputs by running it at least three times.
4. Modify the program of Exercise 4 to compute the area of a triangle. Issue the appropriate prompts for the triangle's base and height, and change the names of the variables appropriately. Then, use the formula $.5 * base * height$ to compute the area. Test the program from an IDLE window

Detecting and Correcting Syntax Errors in Python

Exercises

5 Write and test a program that computes the area of a circle. This program should request a number representing a radius as input from the user. It should use the formula 3.14 * radius ** 2 to compute the area, and output this result suitably labeled.

6 Write and test a program that accepts the user's name (as text) and age (as a number) as input. The program should output a sentence containing the user's name and age.

7 Enter an input statement using the input function at the shell prompt. When the prompt asks you for input, enter a number. Then, attempt to add 1 to that number, observe the results.

8 Enter an input statement using the input function at the shell prompt. When the prompt asks you for input, enter your first name, observe the results.

The software development process

- The systematic planning and organization of a program is referred to as the software development process.
- Various methodologies exist for software development, and one such model is the waterfall model.

The software development process

The waterfall model

The waterfall model consists of several phases:

1. **Customer Request/User Requirements Phase**
2. **Analysis**
3. **Design**
4. **Implementation**
5. **Integration**
6. **Maintenance**

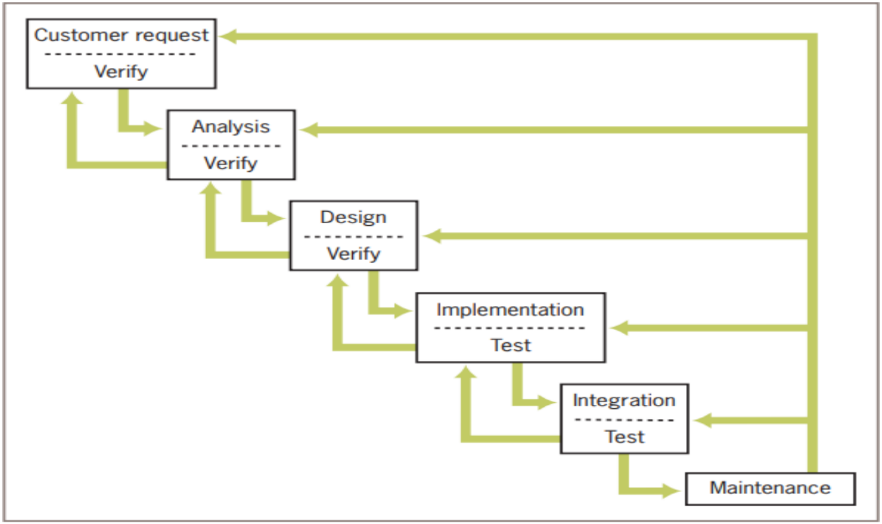
The waterfall model

- **Customer Request/User Requirements Phase:**
  - In this phase, the programmers receive a broad statement of a problem .
  - This step is also called the user requirements phase.
- **Analysis:**
  - The programmers determine what the program will do.
  - This is sometimes viewed as a process of clarifying the specifications for the problem.
- **Design:**
  - The programmers determine how the program will do its task.
- **Implementation:**
  - The programmers write the program.
  - This step is also called the coding phase.
- **Integration:**
  - Large programs have many parts. In the integration phase, these parts are brought together into a smoothly functioning whole, usually not an easy task.

The waterfall model

- **Maintenance:**
  - Programs usually have a long life; a life span of 5 to 15 years is common for software. During this time, requirements change, errors are detected, and minor or major modifications are made.

The waterfall model



The waterfall model

- The figure shows the waterfall model, with each phase's results flowing down to the next. However, if a mistake is found in one phase, developers often need to go back and redo some work from the previous phase. This cycle, including modifications during maintenance, is collectively known as the software development life cycle.
- In modern software development, the process is usually incremental and iterative. This means that analysis and design may produce a basic version or prototype for coding. After some testing, developers may go back to earlier phases to add more details.
- It's important to note that finding and fixing mistakes early is much less expensive in terms of both money and time.

Case Study: Income Tax Calculator

Request:

- The customer requests a program to calculate a person's income tax.

Analysis:

- The analysis involves understanding the problem domain, which, in this case, is tax law.
- Simplifying the tax laws for the case study:
  - All taxpayers face a flat tax rate of 20
  - A standard deduction of $10,000 is allowed.
  - Each dependent adds a deduction of $3,000.
  - Gross income must be entered accurately.
  - The income tax is a decimal number.

Case Study: Income Tax Calculator

Design:

- During the design phase, a clear algorithm is established in pseudocode:
    1. Input gross income and the number of dependents.
    2. Compute taxable income using the formula:
    3. Taxable income = gross income - \$10,000 - (\$3,000 * number of dependents).
    4. Compute income tax using the formula:
    5. Tax = taxable income * 0.20.
    6. Print the calculated tax.

Case Study: Income Tax Calculator

Implementation (Coding):

- The pseudocode is translated into a Python program. Constants are initialized, user inputs are requested, and the income tax is computed and displayed.

Case Study: Income Tax Calculator

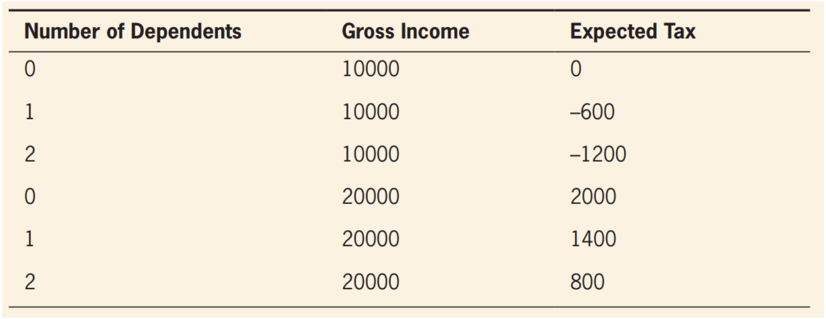
```
# Initialize constants
TAX_RATE = 0.20
STANDARD_DEDUCTION = 10000.0
DEPENDENT_DEDUCTION = 3000.0

# Request user inputs
grossIncome = float(input("Enter the gross income: "))
numDependents = int(input("Enter the number of dependents: "))

# Compute income tax
taxableIncome = grossIncome - STANDARD_DEDUCTION - DEPENDENT_DEDUCTION *
     numDependents
incomeTax = taxableIncome * TAX_RATE

# Display the income tax
print("The income tax is $" + str(incomeTax))
```

Case Study: Income Tax Calculator

Testing:

- Testing is vital for program correctness.
- A test suite is created with various inputs to ensure the program handles different scenarios.
- Expected outputs are compared to actual outputs to identify errors.
- The test suite includes inputs like 0, 1, and 2 for dependents and gross incomes of $10,000 and $20,000.
- The results are tabulated to assess the program's performance.

Case Study: Income Tax Calculator



Number of Dependents	Gross Income	Expected Tax
0	10000	0
1	10000	–600
2	10000	–1200
0	20000	2000
1	20000	1400
2	20000	800

Figure: The test suite for the tax calculator program

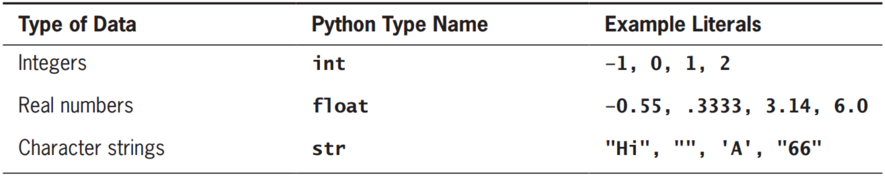
Strings, Assignment, and Comments

Data Types

- A data type consists of a set of values and a set of operations that can be performed on those values.

Literal

- A literal is the way a value of a data type looks to a programmer.
- The programmer can use a literal in a program to mention a data value.



Type of Data	Python Type Name	Example Literals
Integers	`int`	`-1, 0, 1, 2`
Real numbers	`float`	`-0.55, .3333, 3.14, 6.0`
Character strings	`str`	`"Hi", "", 'A', "66"`

Strings, Assignment, and Comments

String Literals

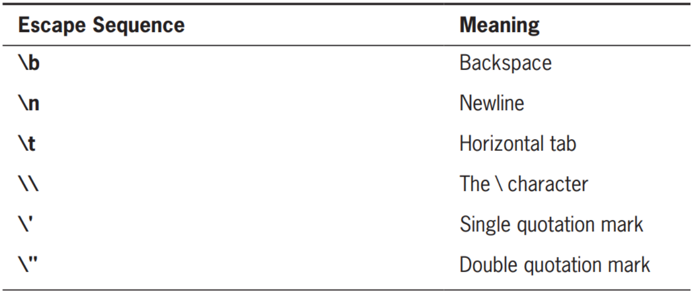
- In Python, a string literal is a sequence of characters enclosed within either single (' ') or double (" ") quotation marks.
- It represents a string value in the code.

```
# Single-line string using single quotes
single_quoted_string = 'Hello, World!'

# Single-line string using double quotes
double_quoted_string = "Python Programming"

# Multiline string using triple single quotes
multiline_single_quoted_string = '''This is a
multiline
string.'''

# Multiline string using triple double quotes
multiline_double_quoted_string = """Another
multiline
string."""
```

Strings, Assignment, and Comments

Escape Sequences

- Escape sequences are sequences of characters in a string that have a special meaning and are preceded by the backslash \ character.



Escape Sequence	**Meaning**
\b	Backspace
\n	Newline
\t	Horizontal tab
\	The \ character
\'	Single quotation mark
\"	Double quotation mark

Strings, Assignment, and Comments

String Concatenation

- In Python, string concatenation involves joining two or more strings to create a new string using the + operator.
- Additionally, the * operator allows you to repeat a string a specified number of times.

```
# String concatenation using the + operator
result = "Hi " + "there, " + "Ken!"
print(result)
# Output: Hi there, Ken!

# Using the * operator to repeat a string
spaces = " " * 10
result_with_spaces = spaces + "Python"
print(result_with_spaces)
# Output:           Python
```

Variables and the Assignment Statement

Variables and the Assignment Statement:

- In programming, a variable is a symbolic name associated with a value, providing a way to remember and use that value later in a program.
- Rules for Variable Names:
  - A variable name must start with a letter or an underscore _.
  - It can contain any combination of letters, digits, or underscores(Cannot start with a digit).
  - Variable names in Python are case-sensitive.

```
# Valid variable names
age = 25
name = "John"
_total_count = 100

# Invalid variable names
123count = 123  # Cannot start with a digit
my-variable = 5  # Cannot contain hyphens
```

Variables and the Assignment Statement

Programmers often use uppercase letters for variables whose values remain constant throughout the program, treating them as symbolic constants.

Assignment Statements:

- An assignment statement is used to bind a variable name to a value.
- The syntax is:

  `<variable_name> = <expression>`
- The Python interpreter evaluates the expression on the right side of the assignment symbol and then associates the variable name on the left side with this value.
- When this association happens for the first time, it is called defining or initializing the variable.

```
# Assigning values to variables
width = 10
height = 5
area = width * height
```

Program Comments and Docstrings

Comments:

- Comments are text within a program that the computer ignores during execution but provide valuable information for programmers.
- In Python, comments can be written using the # symbol, and they extend to the end of the line.

```
RATE = 0.85  # Conversion rate for Canadian to US dollars
```

- In this example, the comment explains the purpose of the variable RATE.

Program Comments and Docstrings

Docstrings:

- Docstrings are special comments enclosed in triple double or single quotes (''' or """) at the beginning of a program or function.
- They provide information about the purpose and usage of the code and are valuable for generating documentation automatically with tools like Sphinx.

```
"""
Program: circle.py
Author: Ken Lambert
Last date modified: 10/10/17
The purpose of this program is to compute the area of a circle.
The input is an integer or floating-point number representing
the radius of the circle. The output is a floating-point number
labeled as the area of the circle.
"""
```

- In this example, the docstring provides information about the program, including the author, the last modification date, and the purpose of the program.

Program Comments and Docstrings

Exercise

1. Let the variable x be “dog” and the variable y be “cat”. Find the values returned by the following operations:
    1. x + y
    2. “the “ + x + “ chases the “ + y
    3. x * 4
2. Write a string that contains your name and address on separate lines using embedded newline characters. Then write the same string literal without the newline characters.

Numeric Data Types and Character Sets

Integers:

- In Python, integers include 0, positive whole numbers, and negative whole numbers.
- Python's int data type has a larger range than some other programming languages,
- limited only by the computer's memory. An example of a large integer operation is given with 2147483647 ** 100.

Numeric Data Types and Character Sets

Floating-Point Numbers:

- Python uses floating-point numbers (float) to represent real numbers.
- Floating-point numbers have a finite range and precision due to computer memory limitations.
- The range of Python's float type is approximately from $2.1x10^{308}$ to $1.0x10^{-308}$, with 16 digits of precision.

Numeric Data Types and Character Sets

Character Sets:

- In Python, character literals are represented as strings.
- The mapping of characters to integer values is defined by character sets, such as ASCII and Unicode.
- ASCII originally had 128 distinct values and later expanded to 256, while Unicode supports 65,536 values.
- Python's ord() and chr() functions convert characters to their numeric ASCII codes and back again, respectively.
    - ord('a') returns the ASCII code for the lowercase letter 'a', which is 97.
    - ord('A') returns the ASCII code for the uppercase letter 'A', which is 65.
    - chr(65) returns the character corresponding to the ASCII code 65, which is 'A'.
    - chr(66) returns the character corresponding to the ASCII code 66, which is 'B'.

Numeric Data Types and Character Sets

```
>>> ord('a')
97
>>> ord('A')
65
>>> chr(65)
A
>>> chr(66)
B
```

Expressions

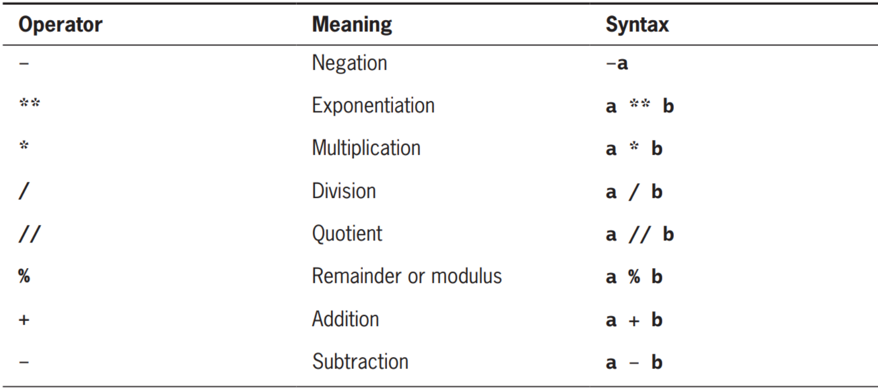
Expressions

- Expressions provide an easy way to perform operations on data values to produce other data values.
    1. Arithmetic Expressions
    2. Mixed-Mode Arithmetic and Type Conversions

Expressions

Arithmetic Expressions:

- An arithmetic expression consists of operands and operators combined in a manner similar to algebra.



Operator	Meaning	Syntax
–	Negation	**–a**
**	Exponentiation	**a ** b**
*	Multiplication	**a * b**
/	Division	**a / b**
//	Quotient	**a // b**
%	Remainder or modulus	**a % b**
+	Addition	**a + b**
–	Subtraction	**a – b**

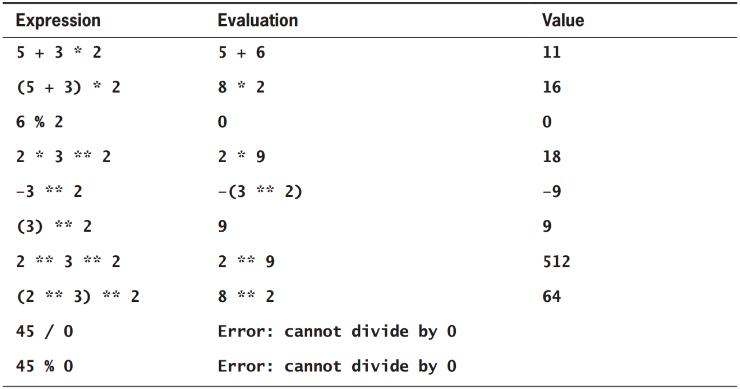
Figure: Arithmetic operators

Expressions

Precedence Rules:

- Exponentiation has the highest precedence and is evaluated first.
- Unary negation is evaluated next, before multiplication, division, and remainder.
- Multiplication, both types of division, and remainder are evaluated before addition and subtraction.
- Addition and subtraction are evaluated before assignment.
- With two exceptions, operations of equal precedence are left associative, so they are evaluated from left to right. Exponentiation and assignment operations are right associative, so consecutive instances of these are evaluated from right to left.
- You can use parentheses to change the order of evaluation.

Expressions



Expression	Evaluation	Value
`5 + 3 * 2`	`5 + 6`	`11`
`(5 + 3) * 2`	`8 * 2`	`16`
`6 % 2`	`0`	`0`
`2 * 3 ** 2`	`2 * 9`	`18`
`-3 ** 2`	`-(3 ** 2)`	`-9`
`(3) ** 2`	`9`	`9`
`2 ** 3 ** 2`	`2 ** 9`	`512`
`(2 ** 3) ** 2`	`8 ** 2`	`64`
`45 / 0`	`Error: cannot divide by 0`	
`45 % 0`	`Error: cannot divide by 0`	

Figure: Some arithmetic expressions and their values

- The quotient operator // produces an integer quotient, whereas the exact division operator / always produces a float.
- Thus, 3//4 produces 0, whereas 3/4 produces .75.

Expressions

Mixed-Mode Arithmetic:

- Involves performing calculations with both integers and floating-point numbers.
- Example:

```
>>> 3.14 * 3 ** 2
28.26
```

- In a binary operation on operands of different numeric types, the less general type (int) is temporarily and automatically converted to the more general type (float) before the operation is performed.

Expressions

Type Conversions:

- A type conversion function is a function with the same name as the data type to which it converts.
- Because the input function returns a string as its value,you must use the function int or float to convert the string to a number before performing arithmetic
    - int(): Converts to an integer (truncating decimals if necessary).
    - float(): Converts to a floating-point number.
    - round(): Rounds a floating-point number to the nearest integer.
    - str(): Converts to a string.
- Implicit Type Conversion (Type Coercion)
    - Python automatically converts data types in certain situations,
        - $x = 5$ # integer
        - $y = 2.0$ # float
        - $z = x + y$ # result is a float (implicit conversion)
- Explicit Type Conversion
    - explicitly convert between data types using built-in functions like ‘int()‘, ‘float()‘, ‘str()‘, etc.

Expressions

Example

```
>>> radius = input("Enter the radius: ")
Enter the radius: 3.2
>>> radius
'3.2'
>>> float(radius)
3.2
>>> float(radius) ** 2 * 3.14
32.153600000000004
```

Another use of type conversion occurs in the construction of strings from numbers and other strings.

```
>>> profit = 1000.55
>>> print('$' + str(profit))
$1000.55
```

Expressions

- In Python, a strong typing approach is employed, where the interpreter rigorously verifies the data types of operands before applying operators.
- If the type of an operand is unsuitable for a specific operation, the interpreter promptly interrupts program execution and issues an error message.

Exercise

1 Let x = 8 and y = 2. find the values of the following expressions:

1. x + y * 3
2. (x + y) * 3
3. x ** y
4. x % y
5. x / 12.0
6. x // 6

2 Let x = 4.66. Find the values of the following expressions:

1. round(x)
2. int(x)

3 Assume that the variable x has the value 55. Use an assignment statement to increment the value of x by 1.

Functions and modules

Functions and modules

- Python includes many useful functions
- Organized in libraries of code called modules

Functions and modules

Calling Functions: Arguments and Return Values

- Function is a chunk of code that can be called by name to perform a task
- Functions require arguments, that is, specific data values, to perform their tasks
- Arguments are also known as parameters
- When a function completes its task, the function may send a result back to the part of the program
- Process of sending a result back to another part of a program is known as returning a value

Functions and modules

Function examples

- round(6.5):

```
result = round(6.5)
print(result)

#Output: 7
```

- abs(4 - 5):

```
result = abs(4 - 5)
print(result)

#Output: 1
```

Functions and modules

Optional arguments and required arguments:

- In Python, a function can have both required and optional arguments. Required arguments are mandatory, while optional arguments have default values and can be omitted.
- round(7.563, 2)

```python
result = round(7.563, 2)
print(result)

#Output: 7.56
```

- help(round)

```python
help(round)

#Output: This will display information about the round function,
    including its signature, parameters, and a brief description.
```

Functions and modules

The math Module

- Functions and other resources are coded in components called modules

```python
import math

# List available functions in the math module
print(dir(math))

# Accessing the value of pi (3.14)
print(math.pi)

# Calculating the square root of 2
print(math.sqrt(2))

# Getting help on the cos function from the math module
help(math.cos)
```

Functions and modules

```
# Importing specific functions/constants from the math module
from math import pi, sqrt

# Using the imported constants and functions
print(pi, sqrt(2))

# Importing all functions/constants from the math module (not
    recommended)
from math import *
```

Functions and modules

Program Format and Structure

- Start with an introductory comment stating the author's name, the purpose of the program, and other relevant information
- Include statements that do the following:
  - Import any modules needed by the program
  - Initialize important variables, suitably commented
  - Prompt the user for input data and save the input data in variables
  - Process the inputs to produce the results
  - Display the results

Functions and modules

Running a Script from a Terminal Command Prompt

- On Windows (Command Prompt):
  1. Open the Command Prompt.
  2. Navigate to the directory where your Python script is located using the cd command. For example:

     ```
     cd path\to\your\script\directory
     ```

  3. Run the Python script using the python command followed by the script's filename:

     ```
     python scriptName.py
     ```

Functions and modules

Running a Script from a Terminal Command Prompt

- On Linux/macOS (Terminal):
  1. Open the Terminal.
  2. Navigate to the directory where your Python script is located using the cd command. For example:

     ```
     cd path/to/your/script/directory
     ```

  3. Run the Python script using the python command followed by the script's filename:

     ```
     python3 scriptName.py
     ```

Exercise

1. You can calculate the surface area of a cube if you know the length of an edge. Write a program that takes the length of an edge (an integer) as input and prints the cube's surface area as output.
2. Five Star Video rents new videos for $3.00 a night, and oldies for $2.00 a night. Write a program that the clerks at Five Star Video can use to calculate the total charge for a customer's video rentals. The program should prompt the user for the number of each type of video and output the total cost.
3. Write a program that takes the radius of a sphere (a floating-point number) as input and outputs the sphere's diameter, circumference, surface area, and volume.
4. An object's momentum is its mass multiplied by its velocity. Write a program that accepts an object's mass (in kilograms) and velocity (in meters per second) as inputs and then outputs its momentum.

Exercise

5 The kinetic energy of a moving object is given by the formula KE=(1/2)mv2, where m is the object's mass and v is its velocity. Modify the program you created in previous exercise so that it prints the object's kinetic energy as well as its momentum.
6 Write a program that calculates and prints the number of minutes in a year.
7 Light travels at 3×10^8 meters per second. A light-year is the distance a light beam travels in one year. Write a program that calculates and displays the value of a light-year'
8 Write a program that takes as input a number of kilometers and prints the corresponding number of nautical miles. Use the following approximations:
    - A kilometer represents 1/10,000 of the distance between the North Pole and the equator.
    - There are 90 degrees, containing 60 minutes of arc each, between the North Pole and the equator.
    - A nautical mile is 1 minute of an arc.

Exercise

9 An employee's total weekly pay equals the hourly wage multiplied by the total number of regular hours plus any overtime pay. Overtime pay equals the total overtime hours multiplied by 1.5 times the hourly wage. Write a program that takes as inputs the hourly wage, total regular hours, and total overtime hours and displays an employee's total weekly pay

Control statements

Control statements

- Control statements in programming languages are used to manage the flow of execution in a program.
- They allow the computer to make decisions and repeat certain actions based on specified conditions.
- There are two main types of control statements:
  1. Selection structures.
  2. Iteration structures.

Control statements

Control statements

- Control statements in programming languages are used to manage the flow of execution in a program.
- They allow the computer to make decisions and repeat certain actions based on specified conditions.
- There are two main types of control statements:
  1. Selection structures.
  2. Iteration structures.

Control statements

Iteration structures.

- Iteration is a general programming concept that refers to the process of repeatedly executing a set of statements or a block of code.
- It's a fundamental control flow mechanism that allows you to perform repetitive tasks in your program.
- There are two main types of iteration:
    1. Definite Iteration
    2. Indefinite Iteration.

Control statements

Definite Iteration.

- Definite iteration involves repeating an action for a known, predefined number of times.
- **Example:** A for loop in Python, where you iterate over a sequence a specific number of times using the range function.

Indefinite Iteration.

- Indefinite iteration involves repeating an action until a certain condition is met or until the program decides to stop.
- **Example:** A while loop in Python, where you continue looping as long as a specified condition remains true.

Control statements

The for Loop

- Syntax of the for Loop:

```
for <variable> in range(<an integer expression>):
    <statement-1>
    .
    .
    <statement-n>
```

  - The first line of a loop is known as the loop header.
  - The integer expression in the header specifies the number of iterations for the loop.
  - The colon (:) signals the end of the loop header.
  - The loop body includes statements beneath the header. It's crucial that these statements are indented consistently in the same column. These statements execute sequentially with each iteration through the loop.

The for Loop

- Example

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

#This loop prints numbers from 0 to 4.
```

- Exponentiation using a for Loop:

```
number = 2
exponent = 3
product = 1
for eachPass in range(exponent):
    product = product * number
    print(product, end=" ")

#Outputs: 2 4 8
```

The for Loop

- To count from a specific starting point, you can provide two arguments to the range function in the loop header.
- The first argument is the lower bound (where you want to start counting), and the second argument is the upper bound (where you want to stop, but not include).

```
for <variable> in range(<lower bound>, <upper bound + 1>):
    <loop body>
```

The for Loop

- Suppose you want to print numbers from 3 to 7 using a for loop. You can use the range function with two arguments:

```
# Explicit lower bound (3) and upper bound (8, not included)
for number in range(3, 8):
    print(number)
```

- In this example:
  - The loop starts at 3 (the lower bound).
  - It continues until 7 (the upper bound + 1, so 8 is not included).
  - The loop prints numbers 3, 4, 5, 6, and 7.

The for Loop

```python
# Python code for calculating factorial

number = 5
fact = 1

for i in range(1, number + 1):
    fact = fact*i

print(fact)
```

The for Loop

Specifying the Steps in the Range

- The range function's third argument is the step value, indicating the interval between numbers.
- This feature allows you to skip some numbers in the range.
- Suppose you want to compute the sum of even numbers between 1 and 10. You can use the range function with a step value of 2:

```python
# Compute the sum of even numbers between 1 and 10
the_sum = 0
for number in range(2, 11, 2):
    the_sum += number

print(the_sum)
```

  - The loop starts at 2 (the first even number in the range).
  - The loop continues with a step of 2, including only even numbers.
  - It stops before 11 (the upper bound + 1).
  - The loop calculates the sum of even numbers (2, 4, 6, 8, 10), resulting in the output 30.

The for Loop

Augmented Assignment:

- Provides shorthand for common operations within loops.
- Examples: +=, -=, *=, /=, %=

```
a = 17
s = "hi"
a += 3 # Equivalent to a = a + 3
a -= 3 # Equivalent to a = a - 3
a *= 3 # Equivalent to a = a * 3
a /= 3 # Equivalent to a = a / 3
a %= 3 # Equivalent to a = a % 3
s += " there" # Equivalent to s = s + " there"
```

The for Loop

Loop Errors - Off-by-One Error:

- Off-by-one errors are common in loop control, often related to incorrectly specifying the upper bound.
- The programmer might intend the following loop to count from 1 through 4, but it counts from 1 through 3:

```python
# Count from 1 through 4, we think
for count in range(1,4):
    print(count)

#Outputs: 1 2 3
```

The for Loop

Traversing the Contents of a Data Sequence

- The for loop is not only limited to counting; it can also be used to traverse the elements in a sequence.
- The range function is used to generate a sequence of numbers.
- Sequences can be of various types, such as lists or strings. Lists are collections of elements, and strings are sequences of characters.

```python
# Example of traversing a list
numbers = [6, 4, 8]
for number in numbers:
    print(number, end=" ")  # Output: 6 4 8

# Example of traversing a string
text = "Hi there!"
for character in text:
    print(character, end=" ")  # Output: H i   t h e r e !
```

The for Loop

Loops That Count Down

- The text introduces the concept of counting down in a loop by specifying a negative step value in the range function.
- It provides an example of a for loop counting down from 10 to 1 and creating a list with the same sequence.

```
# Example of counting down in a loop
for count in range(10, 0, -1):
    print(count, end=" ")  # Output: 10 9 8 7 6 5 4 3 2 1

# Example of creating a list counting down
count_down_list = list(range(10, 0, -1))  # [10, 9, 8, 7, 6, 5, 4,
    3, 2, 1]
```

Exercise

1 Write a loop that prints your name 100 times. Each output should begin on a new line.

2 . Write a loop that prints the first 128 ASCII values followed by the corresponding characters

3 Assume that the variable testString refers to a string. Write a loop that prints each character in this string, followed by its ASCII value.

4 Find the output of the following statements

   1. print(range(10))
   2. print(list(range(10)))
   3. print(list(range(2, 8)))
   4. print(list(range(2, 20, 3)))

Formatting Text for Output

Formatting Text for Output

- In Python, you can format text for output using the % operator or using the {} (format) method.
- However, it's generally recommended to use the newer {} format method as it provides more flexibility and is considered more readable.

Using % operator:

```
name = "John"
age = 25

# Using % operator for string formatting
print("My name is %s and I am %d years old." % (name, age))
```

Using {} (format) method:

```
name = "John"
age = 25

# Using {} format method
print("My name is {} and I am {} years old.".format(name, age))
```

Formatting Text for Output

Using {} (format) method:Using f-string

```
name = "John"
age = 25

# Using f-string
print(f"My name is {name} and I am {age} years old.")
```

Using {} (format) method:

```
name = "John"
age = 25
city = "New York"

# Using indices with {} format method
print("My name is {0}, I am {1} years old, and I live in {2}.".format(
    name, age, city))
```

Formatting Text for Output

Formatting Text for Output
Formatting Strings:

- The % operator is used for formatting, and it works with a format string and a data value.
- For strings, %<field width>s is used. Positive field width right-justifies, negative left-justifies, and if less than or equal to print length, no justification is added.
- Example: "%6s" % "four" right-justifies the string by padding with spaces to the left. Example: "%-6s" % "four" left-justifies by placing spaces to the right.

Formatting Integers:

- For integers, use %d in the format string.

Formatting Floating-Point Numbers:

- For floating-point numbers, the format is %¡field width¿.¡precision¿f.
- Example: "%0.2f" % salary formats a float value with a precision of 2.

Formatting Text for Output

```
# Example of formatting integers in a loop
for exponent in range(7, 11):
    print("%-3d%12d" % (exponent, 10 ** exponent))

#Output
7          10000000
8         100000000
9        1000000000
10      10000000000
```

```
# Example of formatting floating-point numbers
salary = 100.00
print("Your salary is $%0.2f" % salary)

#Output
Your salary is $100.00
```

Selection structure

Selection: if and if-else Statements

- Sometimes, a computer doesn't just follow instructions one after another. It might have to stop and check something before deciding what to do next.
- This checking often involves comparing things, like numbers or conditions.
- The result of this comparison is like saying "yes" (True) or "no" (False).
- So, the computer might pause to figure out what to do based on these comparisons.
    1. If Statement
    2. If-Else Statement
    3. Multi-Way If Statement

Selection structure

If Statement:

- The if statement is a one-way selection statement.
- It executes a block of code only if a specified condition is true.
- Syntax:

```
if condition:
    # code to be executed if the condition is true
```

- Example:

```
x = 10

if x > 5:
    print("x is greater than 5")
```

Selection structure

If-Else Statement:

- The if-else statement is a two-way selection statement.
- It directs the program to execute one block of code if a condition is true and another block if it is false.
- Syntax:

```
if condition:
    # code to be executed if the condition is true
else:
    # code to be executed if the condition is false
```

- Example:

```
y = 3
if y % 2 == 0:
    print("y is even")
else:
    print("y is odd")
```

Selection structure

Multi-Way If Statement:

- A multi-way if statement allows you to test multiple conditions and execute the block of code corresponding to the first true condition.
- Syntax:

```
if condition1:
    # code to be executed if condition1 is true
elif condition2:
    # code to be executed if condition2 is true
elif condition3:
    # code to be executed if condition3 is true
# ... (more elif blocks if needed)
else:
    # code to be executed if none of the conditions is true
```

Selection structure

- Example:

```
grade = 75

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

Selection structure

Logical Operators and Compound Boolean Expressions

- logical operators and compound Boolean expressions are used to handle valid inputs within a specified range
- Handling Valid Inputs:
    - The program aims to process numeric grades within the range of 0 to 100.
    - If the input is above 100, an error message is displayed.
    - If the input is below 0, another error message is displayed.
    - Otherwise, the program proceeds to compute and print the result.

Selection structure

- Simplifying Code with 'or' Operator:
    - The initial code checks two conditions separately and performs the same action for both.
    - The conditions are combined using the or operator to simplify the code.
    - Example:

```
number = int(input("Enter the numeric grade: "))
if number > 100 or number < 0:
    print("Error: grade must be between 100 and 0")
else:
    # The code to compute and print the result goes here
```

Selection structure

- Using 'and' Operator for Range Check:
  - An alternative approach is presented using the and operator to check if the number is within the valid range.
  - Example:

```
number = int(input("Enter the numeric grade: "))
if number >= 0 and number <= 100:
    # The code to compute and print the result goes here
else:
    print("Error: grade must be between 100 and 0")
```

Selection structure

- Logical Operators (and, or, not) in Python:

```
A = True
B = False
print(A and B)  # Output: False
print(A or B)   # Output: True
print(not A)     # Output: False
```

Selection structure

Operator Precedence:

- Example demonstrating how operator precedence affects the evaluation of expressions.

```
A = 2
B = 3
result = A + B * 2 < 10 or B == 2
print(result)  # Output: True
```

Selection structure

Short-Circuit Evaluation in Python:

- Short-circuit evaluation is a feature in Python where the interpreter doesn't need to evaluate all parts of a boolean expression if it can determine the final result early. Specifically:
    - In the expression A and B, if A is False, the whole expression is False, and B doesn't need to be evaluated.
    - In the expression A or B, if A is True, the whole expression is True, and B doesn't need to be evaluated.

Selection structure

- Example:

```
count = int(input("Enter the count: "))
theSum = int(input("Enter the sum: "))

# Short-circuit evaluation in action
if count > 0 and theSum // count > 10:
    print("average > 10")
else:
    print("count = 0 or average <= 10")
```

  - In this example, if the user enters 0 for count, the condition involves potential division by zero (theSum // count).
  - However, because of short-circuit evaluation, if count is 0, the interpreter doesn't need to evaluate the second part of the condition, avoiding a division by zero error.

Selection structure

Exercise

1. Write a loop that counts the number of space characters in a string. Recall that the space character is represented as ' '
2. Assume that the variables x and y refer to strings. Write a code segment that prints these strings in alphabetical order
3. The variables x and y refer to numbers. Write a code segment that prompts the user for an arithmetic operator and prints the value obtained by applying that operator to x and y.

Conditional Iteration: The while Loop

Conditional Iteration: The while Loop

- The while loop is designed for conditional iteration.
- The while loop is an entry-control loop because its condition is tested at the beginning of the loop.
- The statements inside the loop can execute zero or more times based on the condition.

```
while continuation_condition:
    # Statements to execute while the condition is true
```

Infinite Loops:

- The continuation condition in a while loop must eventually become false.
- If it doesn't, the loop will continue indefinitely, causing an error known as an infinite loop.
- To prevent this, at least one statement inside the loop should update a variable affecting the condition.

Conditional Iteration: The while Loop

```
theSum = 0.0
while True:
    data = input("Enter a number or just press enter to quit: ")
    if data == "":
        break
    number = float(data)
    theSum += number
    print("The sum is", theSum)
```

Case Study: Approximating Square Roots

Case Study: Approximating Square Roots

- Request:
    - Write a program that computes square roots.
- Analysis:
    - The program takes a positive floating-point number or integer as input and outputs an estimate of the square root using Newton's method.
    - Python's math.sqrt is used for comparison.
- Design:
    - Newton's method involves successive approximations by taking the average of the current estimate and the number divided by the current estimate.
    - The process continues until the difference between the estimate squared and the original number is within a specified tolerance.
    - The algorithm is implemented with a while True loop for successive approximations.

Case Study: Approximating Square Roots

Implementation (Coding):

```
import math
# Receive the input number from the user
x = float(input("Enter a positive number: "))

# Initialize the tolerance and estimate
tolerance = 0.000001
estimate = 1.0

# Perform the successive approximations
while True:
    estimate = (estimate + x / estimate) / 2
    difference = abs(x - estimate ** 2)
    if difference <= tolerance:
        break

# Output the result
print("The program's estimate:", estimate)
print("Python's estimate: ", math.sqrt(x))
```

Case Study: Approximating Square Roots

Testing:

- Test with perfect squares (e.g., 4, 9).
- Test with inexact square roots (e.g., 2, 3).
- Test with a number between 0 and 1 (e.g., 0.25).
- Adjust the tolerance value during testing to observe the impact on accuracy.

Lazy Evaluation

Lazy Evaluation

- Lazy evaluation is a programming paradigm where the evaluation of an expression is delayed until its value is actually needed.
- In Python, some operations, like those involving generators, exhibit lazy evaluation.
- range function:

```python
# Lazy evaluation using the range function
lazy_range = range(5)

# The range is not immediately created; it's only generated when
    needed
print("Lazy range:", lazy_range)

# Accessing values triggers the actual generation
for value in lazy_range:
    print(value)
```

Lazy Evaluation

- In this example:
  - We create a lazy range object using range(5). The range function in Python is lazy because it doesn't create the entire sequence upfront.
  - When we print "Lazy range," it doesn't display the full range but rather indicates that it is a range object.
  - The actual values are generated and printed when we iterate over the lazy range using a for loop.

Exercise

1. Write a program that accepts the lengths of three sides of a triangle as inputs. The program output should indicate whether or not the triangle is an equilateral triangle or right triangle.
2. Modify the guessing-game program discussed so that the user thinks of a number that the computer must guess. The computer must make no more than the minimum number of guesses.

Exercise

3 The greatest common divisor of two positive integers, A and B, is the largest number that can be evenly divided into both of them. Euclid's algorithm can be used to find the greatest common divisor (GCD) of two positive integers. You can use this algorithm in the following manner:

- Compute the remainder of dividing the larger number by the smaller number.
- Replace the larger number with the smaller number and the smaller number with the remainder.
- Repeat this process until the smaller number is zero
- The larger number at this point is the GCD of A and B
- Write a program that lets the user enter two integers and then prints each step in the process of using the Euclidean algorithm to find their GCD.

Exercise

4 Write a program that receives a series of numbers from the user and allows the user to press the enter key to indicate that he or she is finished providing inputs. After the user presses the enter key, the program should print the sum of the numbers and their average.

5 Write a Python code to check whether a given year is a leap year or not.

6 Write a Python code to print prime numbers between an interval.

7 Write a Python program to find the factorial of a number.

8 Write a program to display the Fibonacci sequence up to n-th term

9 Write a program to check if the number is an Armstrong number or not