

Beginner Programming Fundamentals With Python

Module 2

Input, Processing and Output

Ali Samanipour

Jan 2022

Module Roadmap

- 1 Using Python
- 2 Designing a Program & Problem Solving
- 3 Input, Processing and Output
- 4 Variables & Data Types
- 5 Working with Inputs & Performing Calculation
- 6 Working with Numbers & Strings

Interactive Mode (REPL)

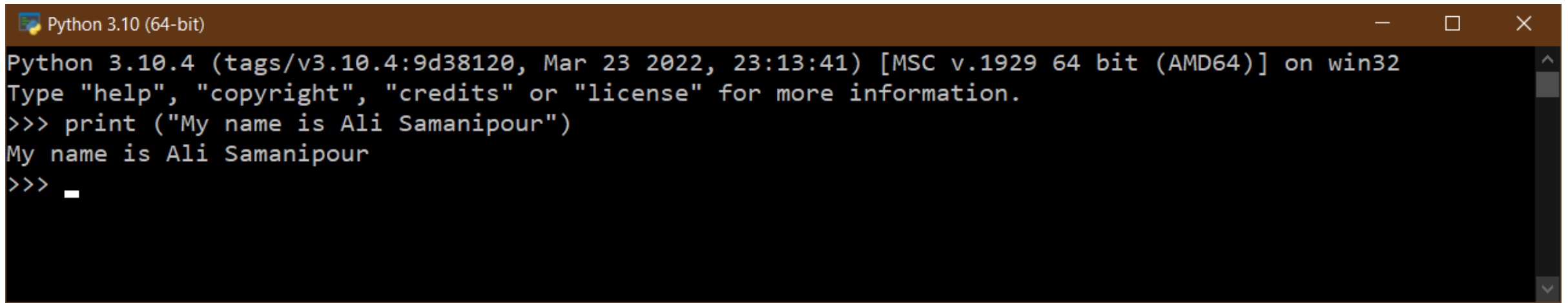
When the Python interpreter is running in interactive mode, it is commonly called the ***Python shell*** or ***REPL***. (useful for testing)



```
Select Python 3.10 (64-bit)
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

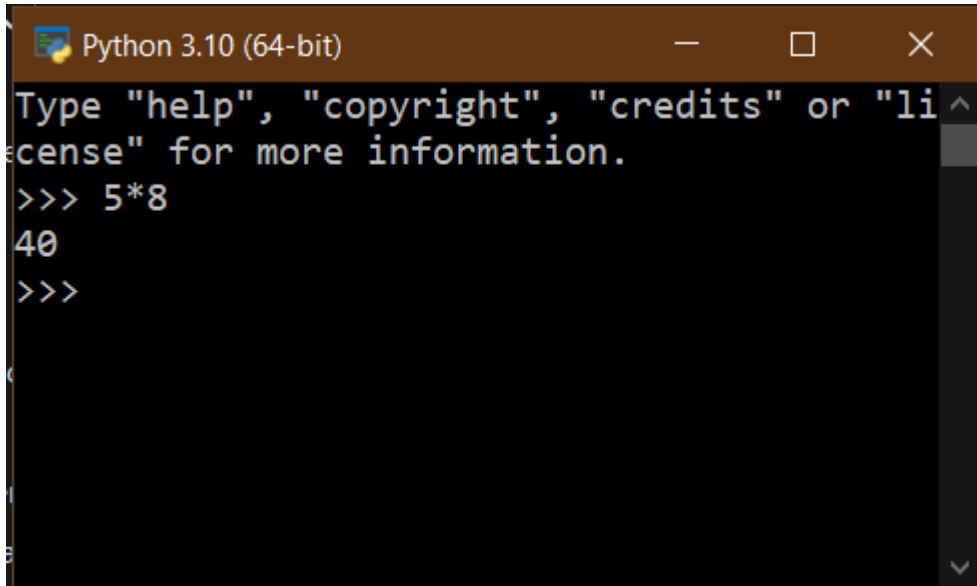
Interactive Mode (REPL)

The `>>>` that you see is a prompt that indicates the interpreter is waiting for you to type a Python statement.

A screenshot of a Python 3.10 (64-bit) REPL window. The window has a title bar with the text "Python 3.10 (64-bit)" and standard window controls. The main area is a dark terminal with white text. It shows the Python version and build information: "Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32". Below this, it says "Type 'help', 'copyright', 'credits' or 'license' for more information." The user has entered the command `>>> print("My name is Ali Samanipour")`, and the interpreter has responded with `My name is Ali Samanipour`. The prompt `>>>` is followed by a cursor.

```
Python 3.10 (64-bit)
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("My name is Ali Samanipour")
My name is Ali Samanipour
>>> _
```

Interactive Mode (REPL) ...



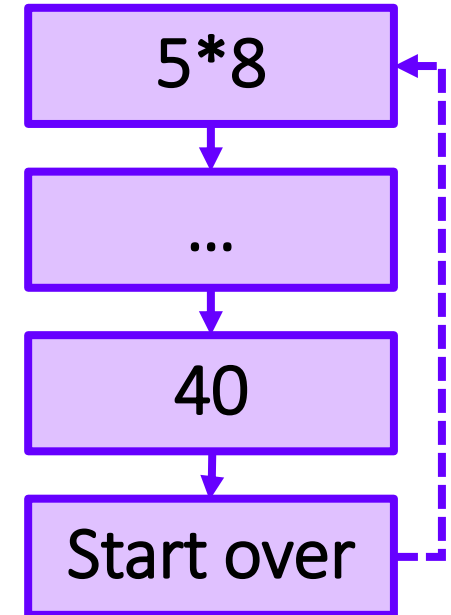
```
Python 3.10 (64-bit)
Type "help", "copyright", "credits" or "license()" for more information.
>>> 5*8
40
>>>
```

Read

Evaluate

Print

Loop



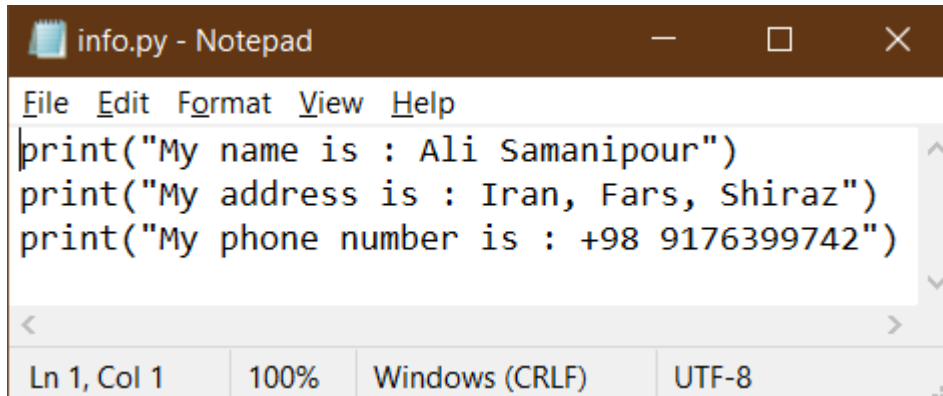
It's useful for practicing and testing. **the statements that you enter in interactive mode are not saved as a program**

Writing Python Programs and Running Them in Script Mode

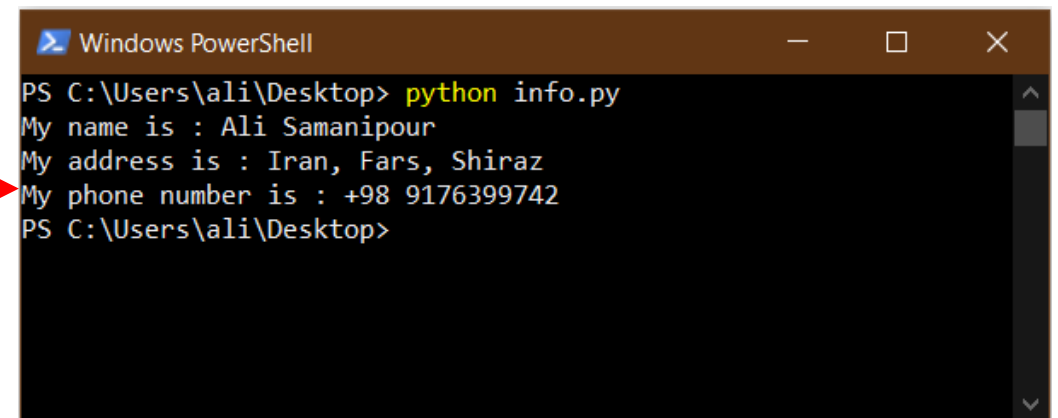
If you want to save a set of Python statements as a program, you save those **statements in a file**. Then, to execute the program, you use the Python interpreter in **script mode**.

1. Write code in plane text using text editor

2. Open shell and execute file



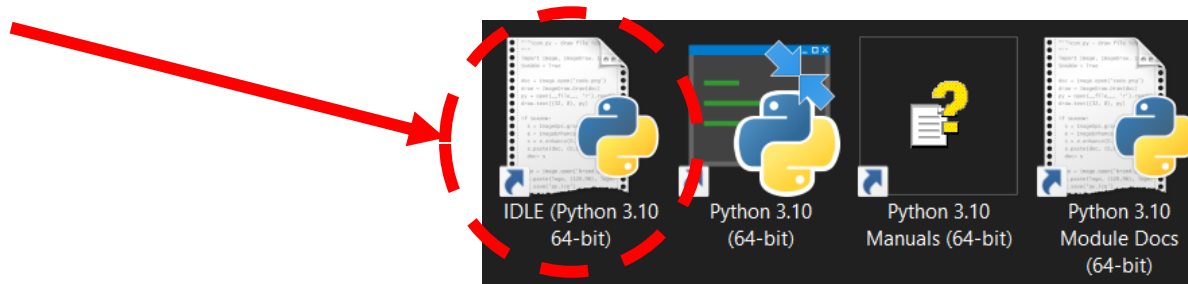
```
File Edit Format View Help
print("My name is : Ali Samanipour")
print("My address is : Iran, Fars, Shiraz")
print("My phone number is : +98 9176399742")
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```



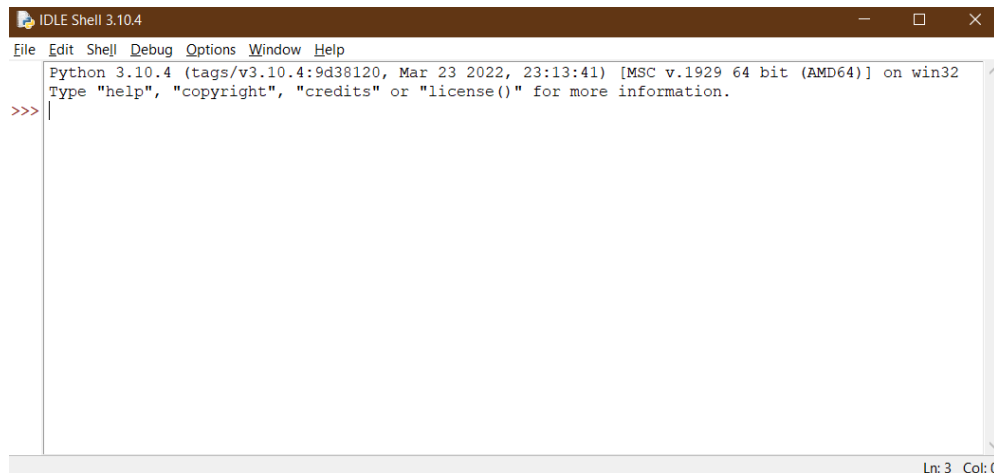
```
Windows PowerShell
PS C:\Users\ali\Desktop> python info.py
My name is : Ali Samanipour
My address is : Iran, Fars, Shiraz
My phone number is : +98 9176399742
PS C:\Users\ali\Desktop>
```

The IDLE Programming Environment

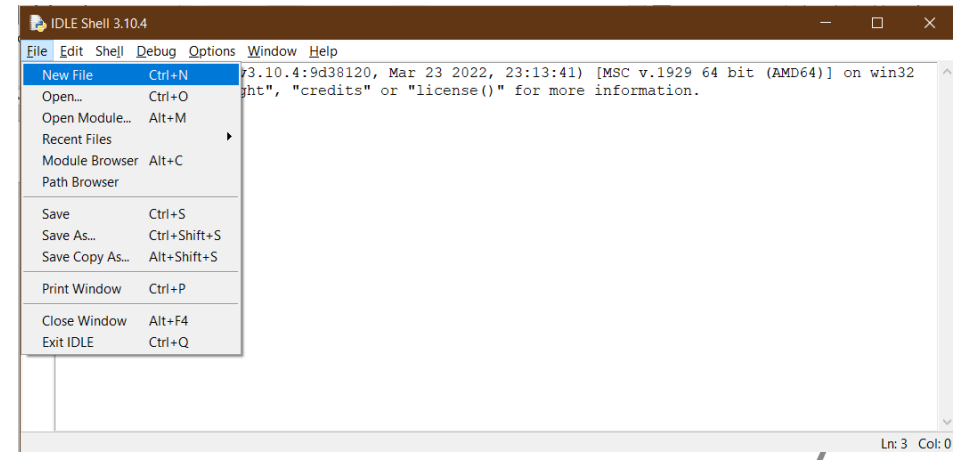
You can use an *integrated development environment*, which is a single program that gives you all of the tools you need to write, execute, and test a program



1. Open IDLE

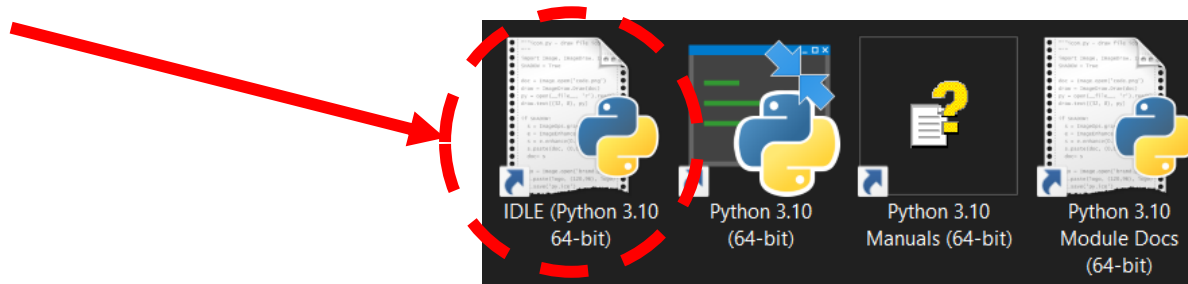


2. Open IDLE Editor



The IDLE Programming Environment ...

You can use an *integrated development environment*, which is a single program that gives you all of the tools you need to write, execute, and test a program



3. Write Code & Save Code

```
test.py - C:/Users/ali/Desktop/test.py (3.10.4)
File Edit Format Run Options Window Help
def display_numbers():
    for i in range(1,10):
        print(i)
display_numbers()
|
```

4. Run The Code

```
test.py - C:/Users/ali/Desktop/test.py (3.10.4)
File Edit Format Run Options Window Help
def display_n
for i in
print
display_numbe
|
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell
```


Advanced IDEs

An **IDE** is a software application that helps programmers develop software code efficiently and provides them all tools they are need to develop a software, including **code editor, syntax highlighting, Intelligent code completion, testing and debugging** and etc. in an integrated environment



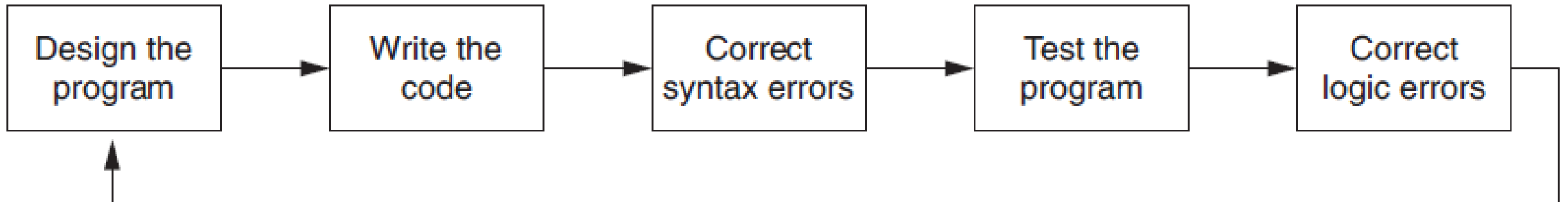
Visual
Studio

Module Roadmap

- 1 Using Python
- 2 Designing a Program & Problem Solving
- 3 Input, Processing and Output
- 4 Variables & Data Types
- 5 Working with Inputs & Performing Calculation
- 6 Working with Numbers & Strings

The Program Development Cycle

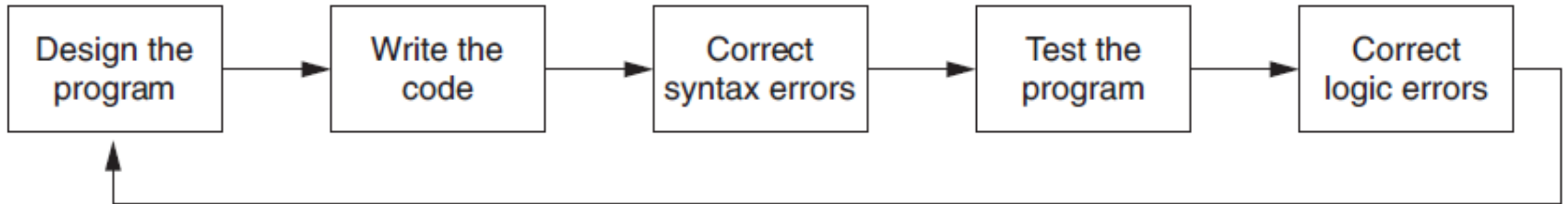
There is much more to creating a program than writing code.
The process of creating a program that works correctly typically requires the five phases shown below



The Program Development Cycle

CONCEPT

Programs must be carefully designed before they are written. During the design process, programmers use tools such as pseudocode and Flowcharts to create models of programs.

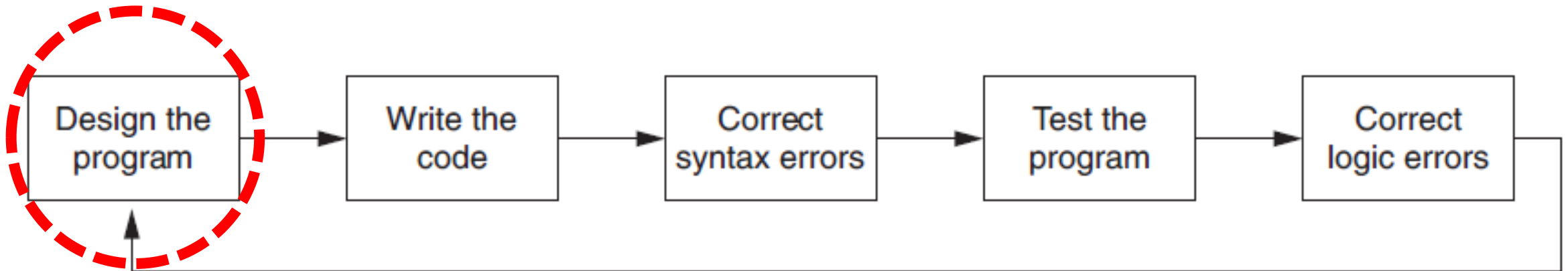


More About the Design Process

CONCEPT

During the design process, programmers use tools such as pseudocode and Flowcharts to create models of programs. This process can be summarized in the following steps:

1. Understand the task that the program is to perform.
2. Determine the steps that must be taken to perform the task.



Understand the Task That the Program Is to Perform

CONCEPT

It is essential that you **understand what a program is supposed to do** before you can determine the steps that the program will perform. Typically, a professional programmer gains this understanding by working directly with the customer.

CONCEPT

The programmer studies the information that was gathered from the customer and creates a list of different software requirements. A **software requirement** is simply a single task that the program must perform in order to satisfy the customer

Determine the Steps That Must Be Taken to Perform the Task

CONCEPT

Once you understand the task that the program will perform, you begin by breaking down the task into a series of steps.

This is similar to the way you would break down a task into a series of steps that another person can follow

What is Algorithm?

suppose someone asks you how to boil water?

1. Pour the desired amount of water into a pot.
2. Put the pot on a stove burner.
3. Turn the burner to high.
4. Watch the water until you see large bubbles rapidly rising. When this happens, the water is boiling.

An **algorithm** is a **lists all of the logical steps** that must be taken by the program.

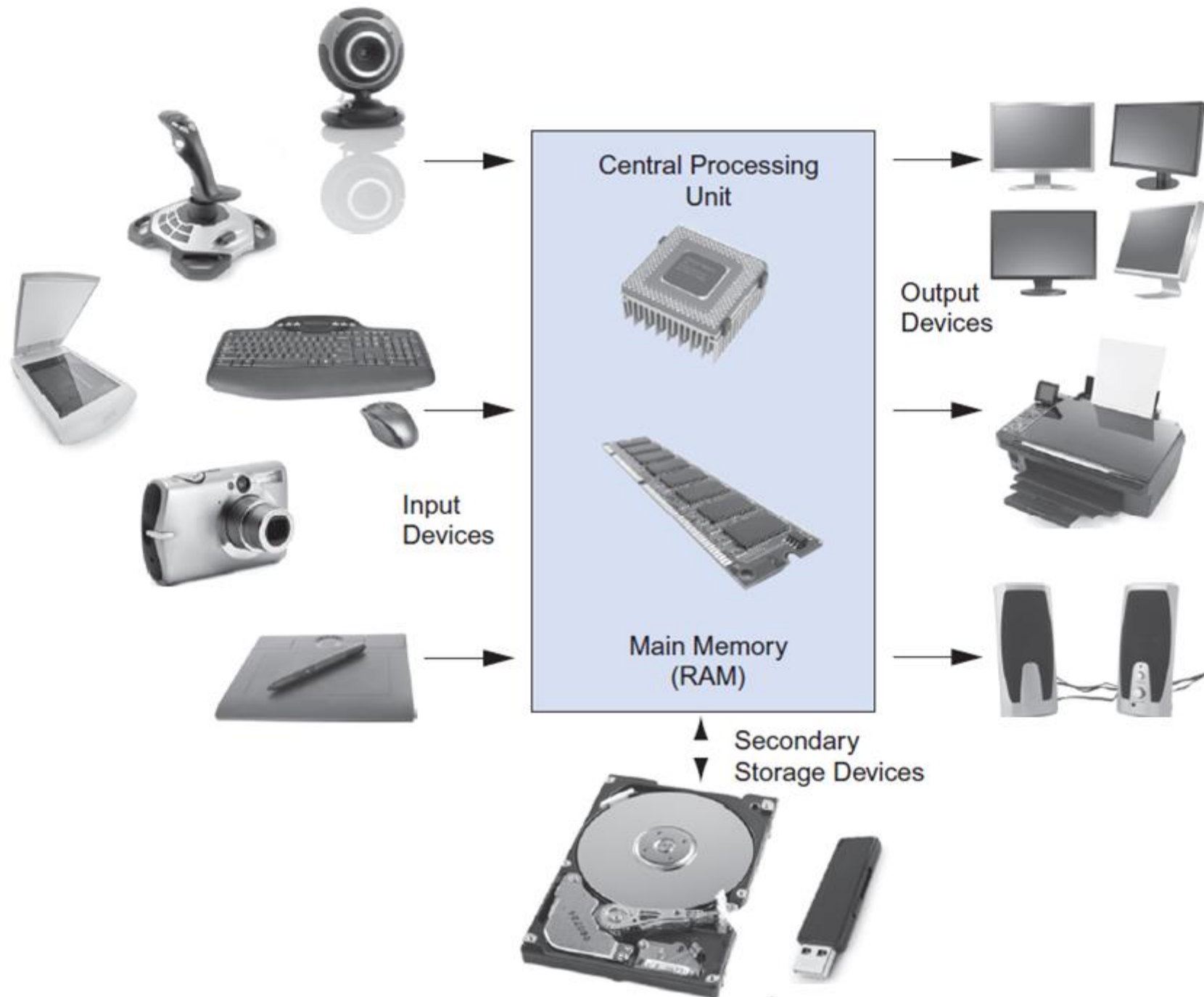
What is Algorithm? ...

CONCEPT

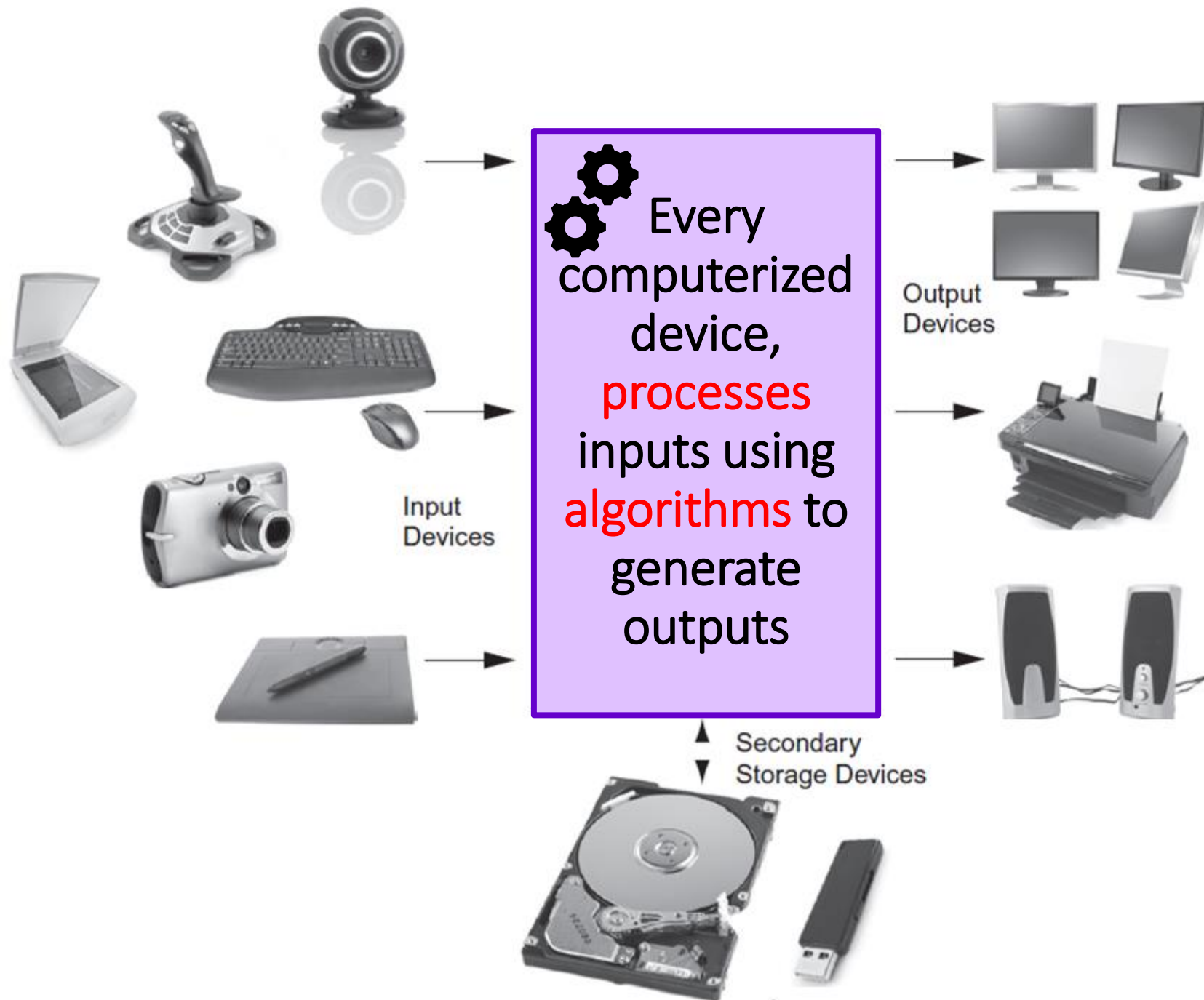
An **algorithm** is a **set of instructions** for **solving a problem** or accomplishing a task.

Tools that help programmers to convert steps of algorithms to executables codes

- **Pseudocode.**
- **Flowcharts.**



Flash Back!
Components of a
computer system



Flash Back!
Components of a
computer system
and Algorithms

Pseudocode

CONCEPT

It is an informal language that has **no syntax rules** and **is not meant to be compiled** or executed. Instead, programmers use *pseudocode* to create models, or “mock-ups,” of programs.

suppose you have been asked to write a pseudocode to calculate and display the gross pay for an hourly paid employee

1. Input the hours worked
2. Input the hourly pay rate
3. Calculate gross pay as hours worked multiplied by pay rate
4. Display the gross pay

Flowcharts

CONCEPT

A *flowchart* is a diagram that graphically depicts the steps that take place in a program.

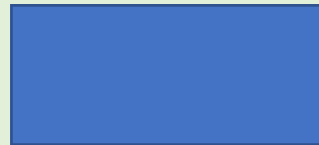
Flowchart Elements



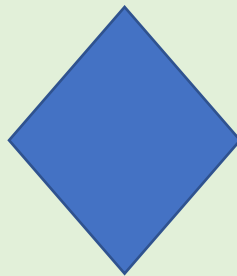
Terminal Symbols, Marks the program's **starting** point and the **end**



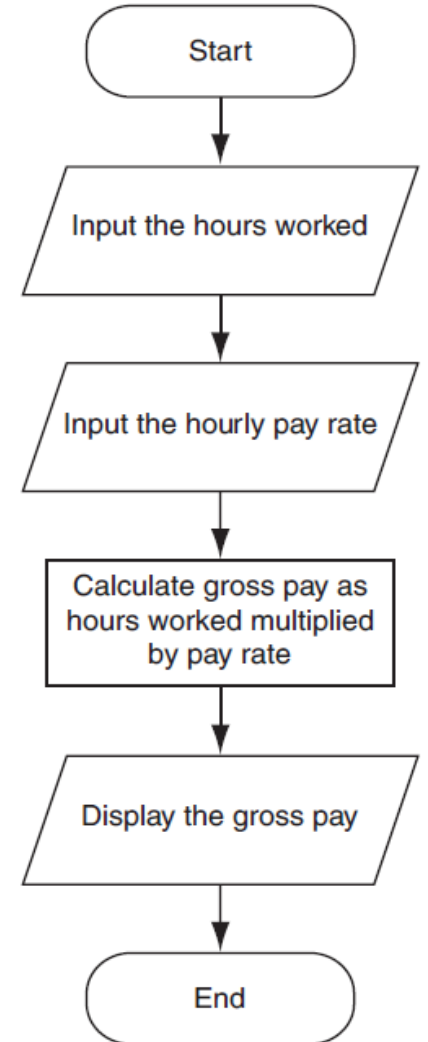
used as **input** symbols and **output** symbols. Steps in which the program reads input or displays output



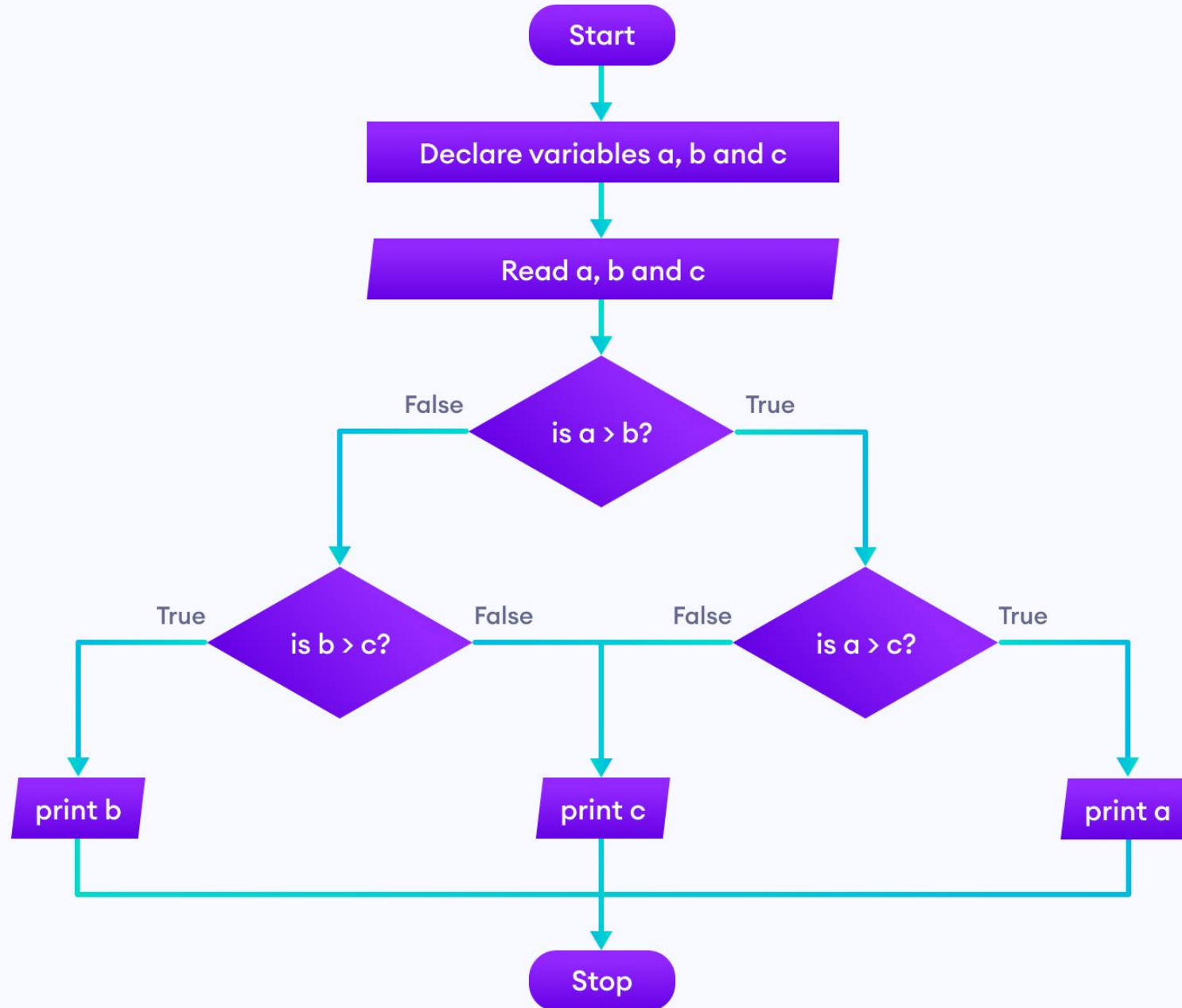
used as **processing** symbols. represent steps in which the program performs some process on data



used as **decision** symbols. represent steps in which the program make a decision based on some conditions



A Flowchart Examples



We use flowcharts to visualize algorithm steps, it helps to understand and design algorithms better

General Problem Solving Techniques

Restate the Problem: In some cases, a problem that looks very difficult may seem easy when stated in a different way or using different terms

Reduce the Problem: reduce the scope of the problem, by either adding or removing constraints, to produce a problem that you do know how to solve.

Divide the Problem: Finding a way to divide a problem into steps or phases can make the problem much easier.

General Problem Solving Techniques

Look for Analogies: An analogy, for our purposes, is a similarity between a current problem and a problem already solved that can be exploited to help solve the current problem.

Experiment: Sometimes the best way to make progress is to try things and observe the results

Start with What You Know: if you never start, you will never finish!

Divide and Conquer Algorithms

Step 1(Divide)

Divide the problem recursively into smaller subproblems.

Step 2(Solve)

Subproblems are solved independently.

Step 3(Combine)

Combine subproblem solutions in order to deduce the answer to the original large problem.

Module Roadmap

- 1 Using Python
- 2 Designing a Program & Problem Solving
- 3 Input, Processing and Output
- 4 Variables & Data Types
- 5 Working with Inputs & Performing Calculation
- 6 Working with Numbers & Strings

Input, Processing, and Output

CONCEPT

Input is data that the program receives. When a program receives data, it usually processes it by performing some operation with it. The result of the operation is sent out of the program as output.

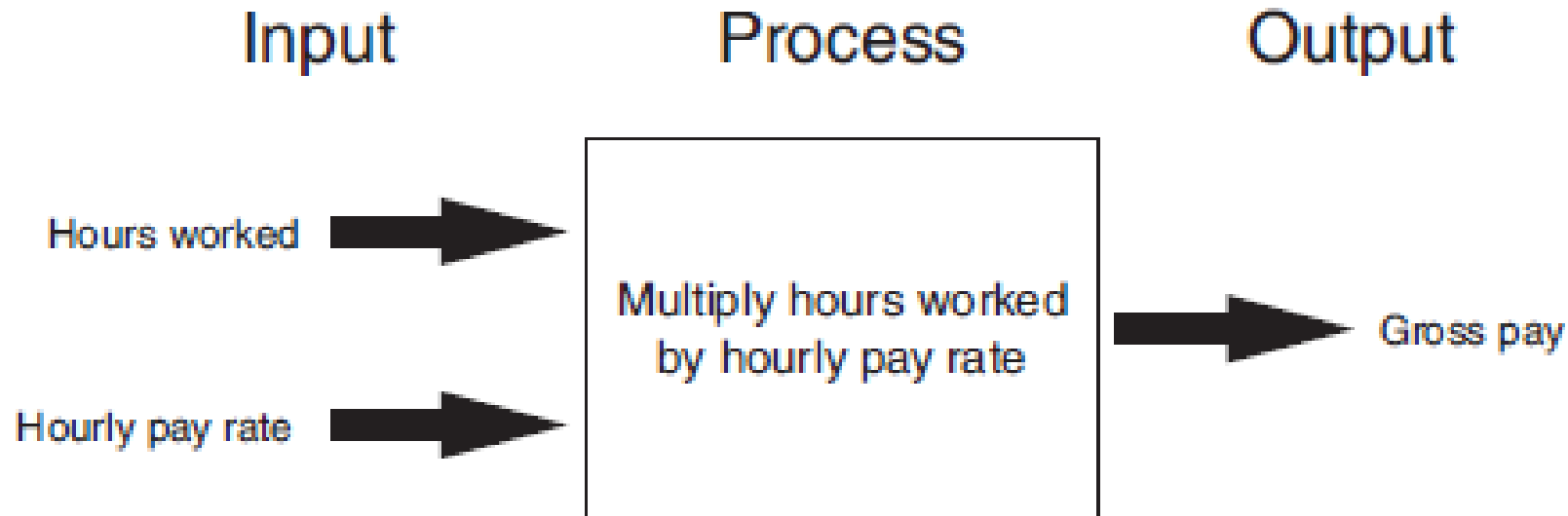
Computer programs typically perform the following three-step process:

- 1. Input is received.**
- 2. Some process is performed on the input.**
- 3. Output is produced.**

Input, Processing, and Output

Computer programs typically perform the following three-step process:

1. Input is received.
2. Some process is performed on the input.
3. Output is produced.



Displaying Output

CONCEPT

A **function** is a piece of prewritten code that performs an operation. Python has numerous **built-in** functions that perform various operations.

You use the **print** function to display output in textual interfaces (like PowerShell or bash)in a Python program.



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

Displaying Output print() Function



```
1 print("Name : Ali Samanipour")  
2 print("Address : Iran, Fars, Shiraz")  
3 print("Phone : +98 9176399742")
```

Module Roadmap

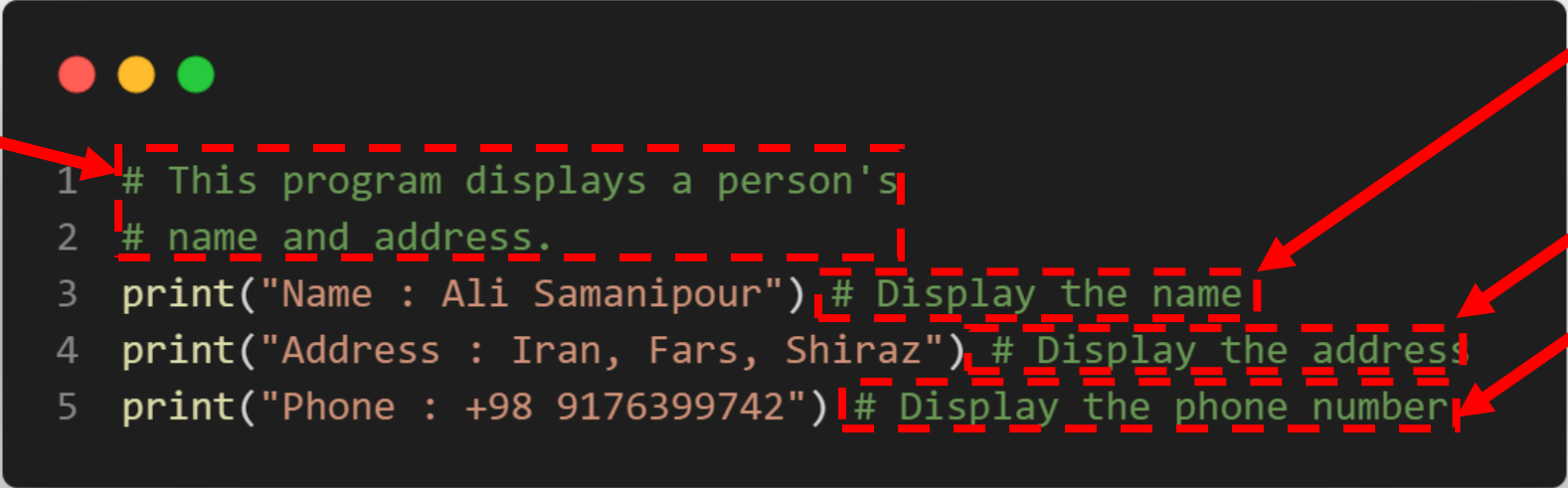
- 1 Using Python
- 2 Designing a Program & Problem Solving
- 3 Input, Processing and Output
- 4 Variables & Data Types
- 5 Working with Inputs & Performing Calculation
- 6 Working with Numbers & Strings

Comment

CONCEPT

Comments are notes of explanation that document lines or sections of a program.

Comments are part of the program, but the **Python interpreter ignores them.**



```
1 # This program displays a person's
2 # name and address.
3 print("Name : Ali Samanipour") # Display the name
4 print("Address : Iran, Fars, Shiraz") # Display the address
5 print("Phone : +98 9176399742") # Display the phone number
```


Variables

CONCEPT

A variable is a name that **represents a storage location** in the computer's memory.

When a variable represents a value in the computer's memory, we say that the variable ***references*** the value

Creating Variables

CONCEPT

You use an assignment statement to create a variable and make it reference a piece of data

variable = expression



```
1 age = 25
```

age

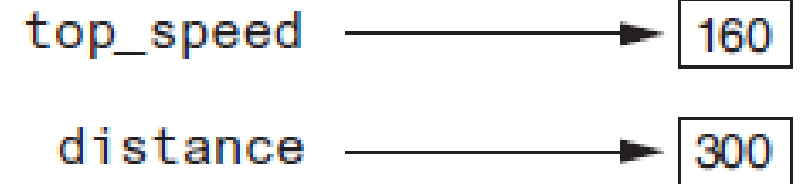


25

Creating Variables ...



```
1 # Create two variables: top_speed and distance.
2 top_speed = 160
3 distance = 300
4 # Display the values referenced by the variables.
5 print('The top speed is')
6 print(top_speed)
7 print('The distance traveled is')
8 print(distance)
```



Variable Naming Rules

You **cannot** use one of Python's **keywords** as a variable name.

A variable name **cannot contain spaces**

Uppercase and **lowercase** characters are distinct

After the first character you may use the letters a through z or A through Z, the digits 0 through 9, or underscores

The first character must be one of the letters a through z, A through Z, or an underscore character (_).

Variable Reassignment



```
1 # This program demonstrates variable reassignment.
2 # Assign a value to the dollars variable.
3 dollars = 2.75
4 print('I have', dollars, 'in my account.')
5
6 # Reassign dollars so it references
7 # a different value.
8 dollars = 99.95
9 print('But now I have', dollars, 'in my account!')
```

The dollars variable after line 3 executes.

dollars → 2.75

The dollars variable after line 8 executes.

dollars → 2.75
dollars → 99.95

Primitive Datatypes

-5

-4.97

True

False

'Ali'

'Number is:'

2

1.97

Integer

Floats

Numbers

Boolean

String

Complex Data Types(Dictionaries, Classes, ...)

Numbers

Integer

-5

4

As big(small) as supported
by memory and OS

cast other type to integer
with `int()`

Float

1.58

-0.8559

As big(small) as supported
by memory and OS

cast other type to integer
with `float()`

Write numbers with more readability: 100_258_012.587

Module Roadmap

- 1 Using Python
- 2 Designing a Program & Problem Solving
- 3 Input, Processing and Output
- 4 Variables & Data Types
- 5 Working with Inputs & Performing Calculation
- 6 Working with Numbers & Strings

Reading Input from the Keyboard


CONCEPT

Programs commonly need to read *input* typed by the *user* on the keyboard

variable = input(prompt)

```
1 # Get the user's first name.
2 first_name = input('Enter your first name: ')
3
4 # Get the user's last name.
5 last_name = input('Enter your last name: ')
6
7 # Print a greeting to the user.
8 print('Hello', first_name, last_name)
```

Reading Numbers with the input Function



```
1  # Get the user's name, age, and income.
2  name = input('What is your name? ')
3  age = int(input('What is your age? '))
4  income = float(input('What is your income? '))
5
6  # Display the data.
7  print('Here is the data you entered:')
8  print('Name:', name)
9  print('Age:', age)
10 print('Income:', income)
```

Python *Mathematical* Operators

CONCEPT

Python has numerous **operators** that can be used to perform mathematical calculations.

+

Addition: Adds two numbers

$9 + 5 \rightarrow 14$

-

Subtraction: Subtracts one number from another

$9 - 5 \rightarrow 4$

*

Multiplication: Multiplies one number by another

$9 * 5 \rightarrow 45$

/

Division Divides: one number by another and gives the result as a floating-point number

$9 / 5 \rightarrow 1.8$

//

Integer division: Divides one number by another and gives the result as a whole number

$9 // 5 \rightarrow 1$

**

Exponent: Raises a number to a power

$9 ** 5 \rightarrow 59049$

%

Remainder: Divides one number by another and gives the remainder

$9 \% 5 \rightarrow 4$

Operator Precedence

The precedence of the math operators, from highest to lowest, are:

1. Exponentiation: **
2. Multiplication, division, and remainder: * / // %
3. Addition and subtraction: + -

Operations that are enclosed in parentheses are performed first. Then, when two operators share an operand, the operator with the higher *precedence* is applied first

Example: Time Converter Program

```
1  # Get a number of seconds from the user.
2  total_seconds = float(input('Enter a number of seconds: '))
3
4  # Get the number of hours.
5  hours = total_seconds // 3600
6
7  # Get the number of remaining minutes.
8  minutes = (total_seconds // 60) % 60
9
10 # Get the number of remaining seconds.
11 seconds = total_seconds % 60
12
13 # Display the results.
14 print('Here is the time in hours, minutes, and seconds:')
15 print('Hours:', hours)
16 print('Minutes:', minutes)
17 print('Seconds:', seconds)
```

Example: Future Value Program (Question)

Suppose you want to deposit a certain amount of money into a savings account and leave it alone to draw interest for the next 10 years. At the end of 10 years, you would like to have \$10,000 in the account. How much do you need to deposit today to make that happen? You can use the following formula to find out:

$$P = \frac{F}{(1+r)^n}$$


The terms in the formula are as follows:

- P is the present value, or the amount that you need to deposit today.
- F is the future value that you want in the account. (In this case, F is \$10,000.)
- r is the annual interest rate.
- n is the number of years that you plan to let the money sit in the account.

Example: Future Value Program (Algorithm – pseudocode)

- 1. Get the desired future value.*
- 2. Get the annual interest rate.*
- 3. Get the number of years that the money will sit in the account.*
- 4. Calculate the amount that will have to be deposited.*
- 5. Display the result of the calculation in step 4.*

Example: Future Value Program (Source Code)



```
1  # Get the desired future value.
2  future_value = float(input('Enter the desired future value: '))
3
4  # Get the annual interest rate.
5  rate = float(input('Enter the annual interest rate: '))
6
7  # Get the number of years that the money will appreciate.
8  years = int(input(
9      'Enter the number of years the money will grow: '))
10
11 # Calculate the amount needed to deposit.
12 present_value = future_value / (1.0 + rate)**years
13
14 # Display the amount needed to deposit.
15 print('You will need to deposit this amount:', present_value)
```


Mixed-Type Expressions and Data Type Conversion

CONCEPT

When you perform a math operation on two operands, the data type of the result will depend on the data type of the operands

```
1 # This program shows how python follows these rules when evaluating mathematical expressions
2 int_var1 = 5
3 int_var2 = -8
4 float_var1 = 2.138
5 float_var2 = -0.257
6
7 # When an operation is performed on two int values, the result will be an int.
8 print("Summing tow int :", int_var1 + int_var2)
9
10 # When an operation is performed on two float values, the result will be a float.
11 print("Summing tow float :", float_var1 + float_var2)
12
13 # When an operation is performed on an int and a float, the int value will be
14 print("Summing one int and float :", int_var1 + float_var2)
```

Module Roadmap

- 1 Using Python
- 2 Designing a Program & Problem Solving
- 3 Input, Processing and Output
- 4 Variables & Data Types
- 5 Working with Inputs & Performing Calculation
- 6 Working with Numbers & Strings

Strings and Literals

CONCEPT

In programming terms, a sequence of characters that is used as data is called a string. When a string appears in the actual code of a program, it is called a *string literal*.

“single line text”

‘single line text’

“”multiple line
text””

String Concatenation

String concatenation is the appending of one string to the end of another.

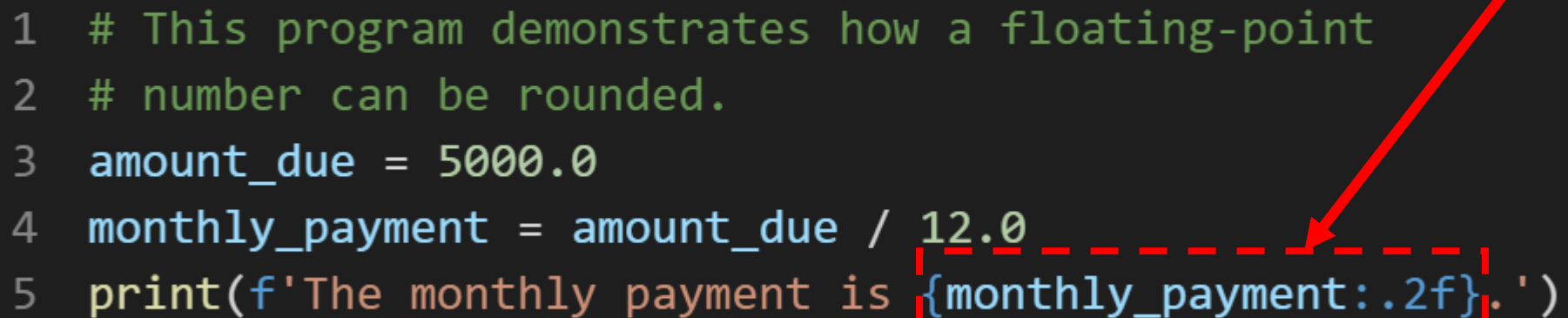


```
1  # This program demonstrates string concatenation.
2  first_name = input('Enter your first name: ')
3  last_name = input('Enter your last name: ')
4
5  # Combine the names with a space between them.
6  full_name = first_name + ' ' + last_name
7
8  # Display the user's full name.
9  print('Your full name is ' + full_name)
```

Displaying Formatted Output with F-strings

CONCEPT

F-strings are a special type of string literal that allow you to format Values in a variety of ways



```
1 # This program demonstrates how a floating-point
2 # number can be rounded.
3 amount_due = 5000.0
4 monthly_payment = amount_due / 12.0
5 print(f'The monthly payment is {monthly_payment:.2f}.')
```

Displaying Formatted Output with F-strings



```
1  # This program demonstrates how to align strings.
2  string_var1 = 'left align string'
3  string_var2 = 'center align string'
4  string_var3 = 'right align string'
5
6  # Display the texts.
7
8  # < Designator Left-aligns the value
9  print(f'***{string_var1:<50}***')
10 # ^ Designator Center-aligns the value
11 print(f'***{string_var2:^50}***')
12 # > Designator Right-aligns the value
13 print(f'***{string_var3:>50}***')
```


Escape Characters

CONCEPT

An ***escape character*** is a special character that is preceded with a backslash (\), appearing inside a string literal.

When a string literal that contains escape characters is printed, the escape characters are treated as special commands that are embedded in the string.

Using Escape Characters



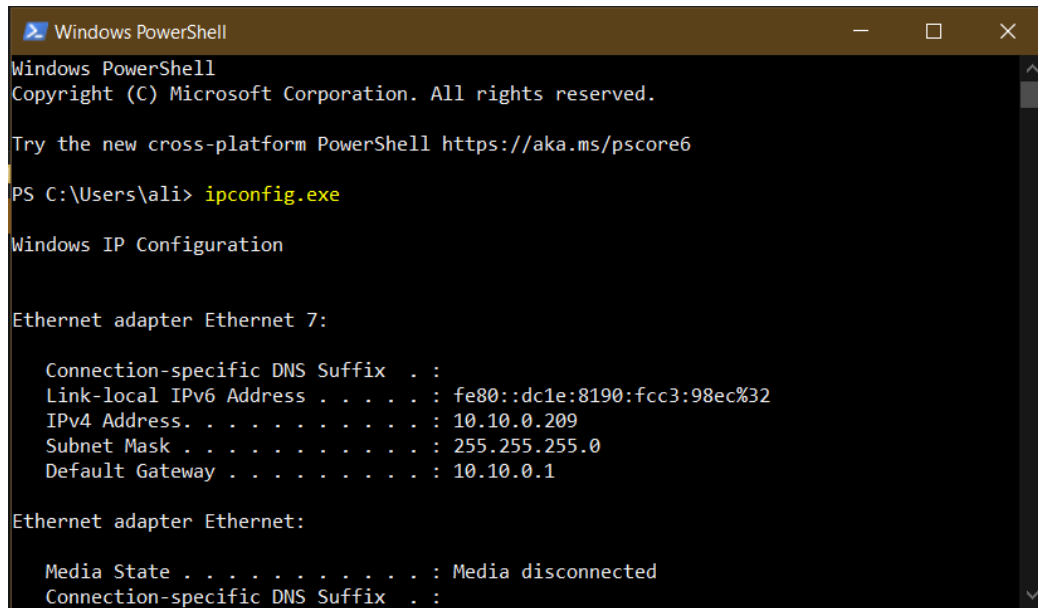
```
1  #this program illustrate functions of escape characters
2
3  # \n Causes output to be advanced to the next line.
4  print("First line \n Second line \n Third line")
5
6  # \t Causes output to skip over to the next horizontal tab position
7  print('Sat\tSun\tMon\tTues\tWed\tThur')
8
9  # \' and \" escape characters to display quotation marks
10 print("Your assignment is to read \"Hafez\" by tomorrow.")
11 print('I\'m ready to begin.')
12
13 # \\ Causes a backslash character to be printed
14 print('The path is C:\\temp\\data.')
```


GUI vs CLI

CONCEPT

GUI lets a user interact with the device/system with the help of graphical elements, like windows, menus, icons, etc. The **CLI**, on the other hand, lets a user interact with their device/system with the help of various commands

CLI



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ali> ipconfig.exe

Windows IP Configuration

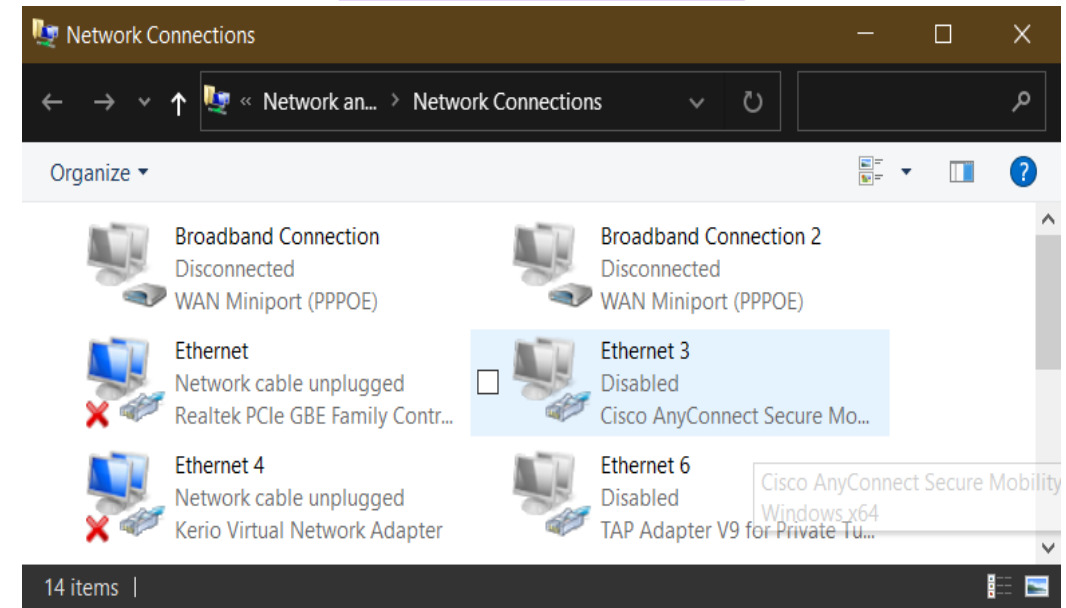
Ethernet adapter Ethernet 7:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::dc1e:8190:fcc3:98ec%32
    IPv4 Address. . . . . : 10.10.0.209
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 10.10.0.1

Ethernet adapter Ethernet:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :
```

GUI



Named Constants

CONCEPT

A ***named constant*** is a name that represents a special value, such as a magic number.

named constant is written in all **uppercase** letters. This is a standard practice in most programming languages because it makes named constants easily distinguishable from regular variables.

Benefit of Using Named Constants



```
1 radius = float(input("please input circle radius "))
2 circle_area = 3.14 * (radius**2)
3 print(f'Circle area is : {circle_area:.3f}')
```



```
1 PI_VALUE = 3.14
2 radius = float(input("please input circle radius "))
3 circle_area = PI_VALUE * (radius**2)
4 print(f'Circle area is : {circle_area:.3f}')
```

Appendix: Python Documentation

The screenshot shows a web browser window displaying the Python 3.10.2 documentation page. The browser's address bar shows the URL `https://docs.python.org/3/`. The page has a dark-themed header with navigation links like 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Tools', and 'Help'. Below the header, there's a sidebar on the left with sections: 'Download' (with a link to download documents), 'Docs by version' (listing Python versions from 3.11 down to 2.7, with 3.10.2 selected), and 'Other resources' (including PEP Index, Beginner's Guide, Book List, Audio/Visual Talks, and Python Developer's Guide). The main content area is titled 'Python 3.10.2 documentation' and includes a welcome message. It lists 'Parts of the documentation' with links and descriptions for: 'What's new in Python 3.10?' (or all 'What's new' documents since 2.0), 'Tutorial' (start here), 'Library Reference' (keep this under your pillow), 'Language Reference' (describes syntax and language elements), 'Python Setup and Usage' (how to use Python on different platforms), 'Python HOWTOs' (in-depth documents on specific topics), 'Installing Python Modules' (installing from the Python Package Index & other sources), 'Distributing Python Modules' (publishing modules for installation by others), 'Extending and Embedding' (tutorial for C/C++ programmers), 'Python/C API' (reference for C/C++ programmers), and 'FAQs' (frequently asked questions (with answers!)). At the bottom, it lists 'Indices and tables' with links to 'Global Module Index' (quick access to all modules) and 'General Index'. A search bar is located at the bottom right of the main content area, with the text 'Search page' and 'search this documentation'.

Python 3.10.2 documentation

Welcome! This is the official documentation for Python 3.10.2.

Parts of the documentation:

- [What's new in Python 3.10?](#)
or all "What's new" documents since 2.0
- [Tutorial](#)
start here
- [Library Reference](#)
keep this under your pillow
- [Language Reference](#)
describes syntax and language elements
- [Python Setup and Usage](#)
how to use Python on different platforms
- [Python HOWTOs](#)
in-depth documents on specific topics
- [Installing Python Modules](#)
installing from the Python Package Index & other sources
- [Distributing Python Modules](#)
publishing modules for installation by others
- [Extending and Embedding](#)
tutorial for C/C++ programmers
- [Python/C API](#)
reference for C/C++ programmers
- [FAQs](#)
frequently asked questions (with answers!)

Indices and tables:

- [Global Module Index](#)
quick access to all modules
- [General Index](#)

Search page
search this documentation

Appendix: Python Coding style

The screenshot shows a web browser window displaying the Python.org website. The browser's address bar shows the URL `https://www.python.org/dev/peps/pep-0008/`. The website's navigation bar includes links for Python, PSF, Docs, PyPI, Jobs, and Community. Below this, there is a search bar and a 'Donate' button. The main content area features a sidebar on the left with a link to 'Tweets by @ThePSF' and a blue box containing text about the Python Software Foundation. The main content area displays the title 'PEP 8 -- Style Guide for Python Code' and a table with details about the PEP.

[Tweets by @ThePSF](#)

The PSF
The Python Software Foundation is the organization behind Python. Become a member of the PSF and help advance the software and our mission.

[Python](#) >>> [Python Developer's Guide](#) >>> [PEP Index](#) >>> [PEP 8 -- Style Guide for Python Code](#)

PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Ali Samanipour
[linkedin.com/in/Samanipour](https://www.linkedin.com/in/Samanipour)