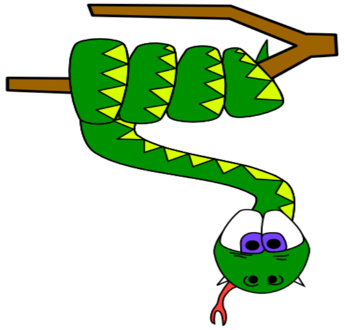


# Computer Programming I

Ishaya Gambo

## Lecture Goals

- To understand the basic structure and syntax of Python programming language
- To write your own simple Python scripts.
- To serve as the starting point for more advanced training on Python coding

# Computers and Programming

- A computer is composed of a Central Processing Unit (abbreviated CPU), memory, and Input/Output (abbreviated I/O) devices. A screen is an output device.

- A computer is just a machine (the *hardware*) for executing programs (the *software*)
  - Hence, the software rules the hardware!

- The process of creating software is called *programming*, and it is the focus of this course

- Virtually, anyone can learn how to program computers
  - It requires only some grit!

## Why Learn Programming?

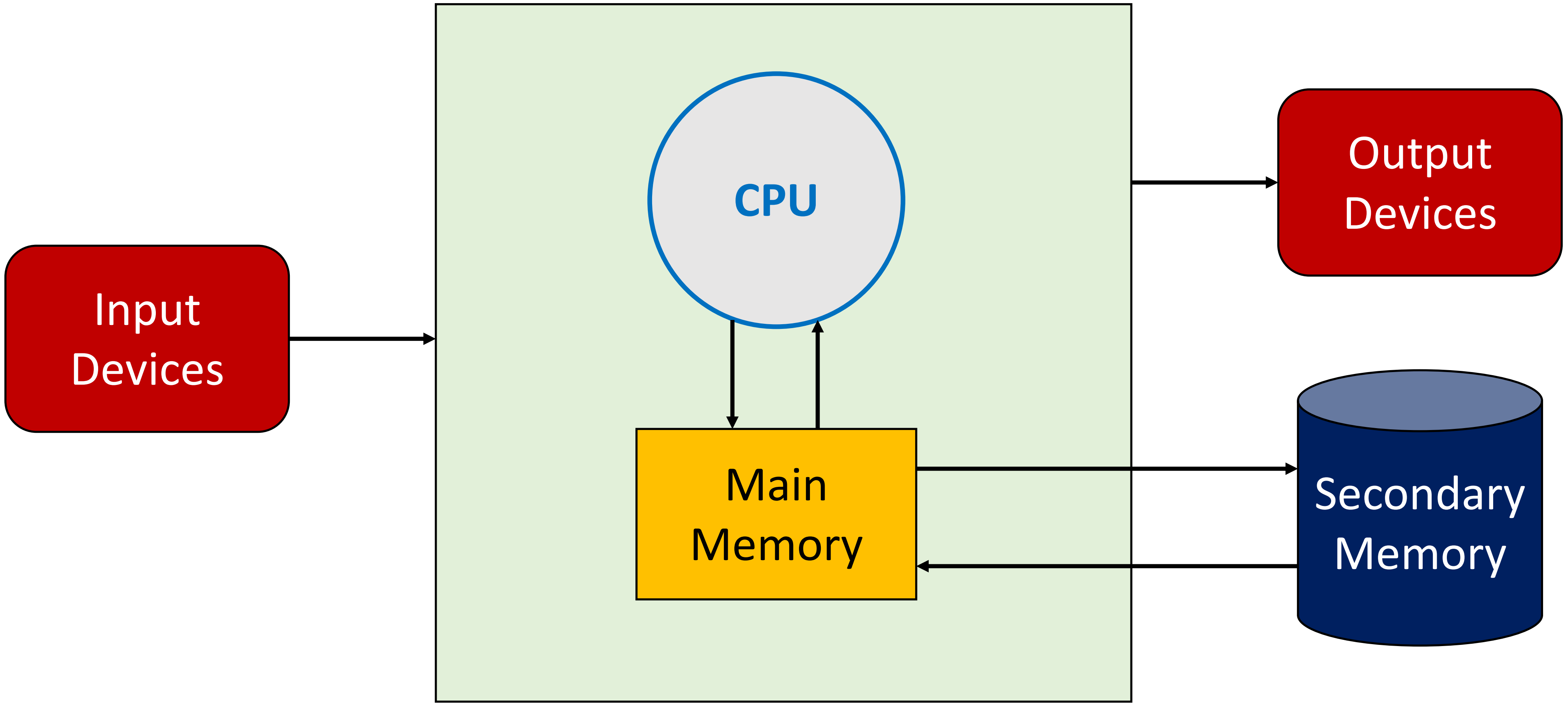
- Computers have become commonplace in our modern life
  - Applying ideas in different fields requires programming
- Programming can be loads of fun!
  - It is an intellectually engaging activity that allows you to express yourself through remarkably beautiful constructs and structures
- Programming develops valuable problem-solving skills, especially ones that pertain to analysis, design and implementation
- Programmers are in great demand!

## Hardware Basics

- To be a successful programmer, you need to know some details of how computers work

- For instance, understanding the basics of hardware will help you analyze the *performance* (or *efficiency*) of any of your programs
  - Will the data of your program fit in *memory*?
    - If not, how would that impact the performance of your program?
  - Is your program *CPU-bound* or *IO-Bound*?
    - If CPU-bound, how powerful is your CPU?
    - If IO-bound, how big is your disk or network bandwidth?

# Functional View of a Computer

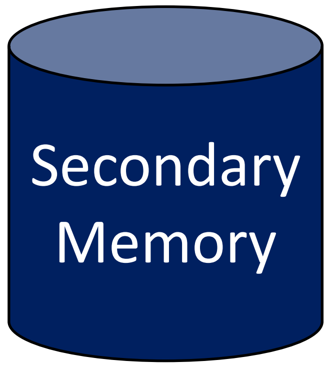


# Functional View of a Computer

- The secondary memory is where your *saved program and data* reside

- It is a *non-volatile* storage
  - I.e., when the power is turned off,
    your program and data will NOT be lost

**E.g., Hard Disk**

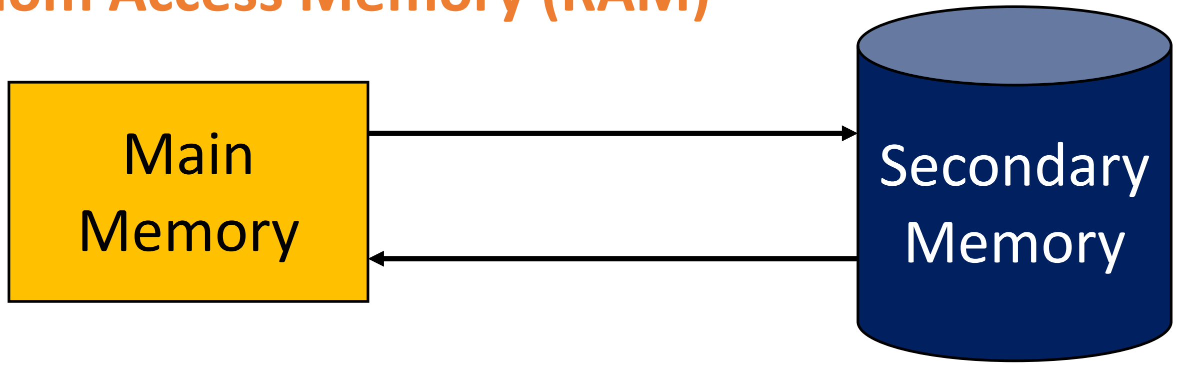


# Functional View of a Computer

- The main memory is much faster (but more expensive) than the secondary one, however, it is *volatile*

- Your program and data are *copied* from secondary memory to main memory for efficiency reasons

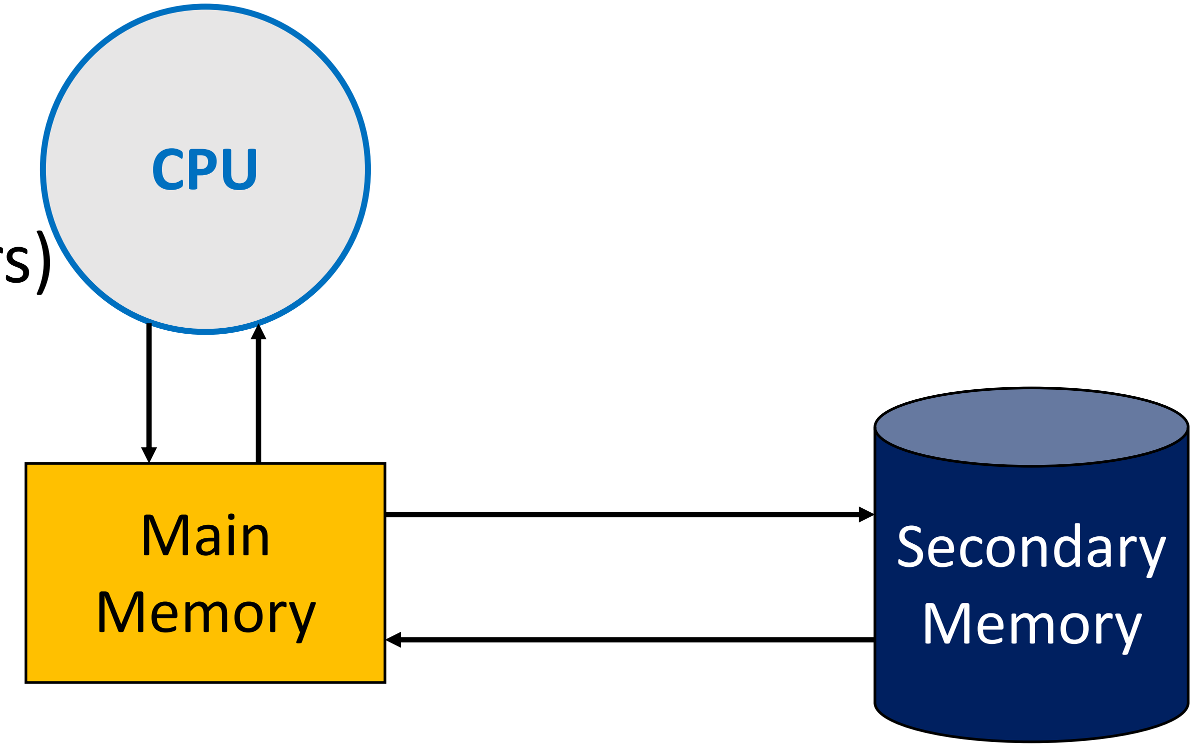
**E.g., Random Access Memory (RAM)**



# Functional View of a Computer

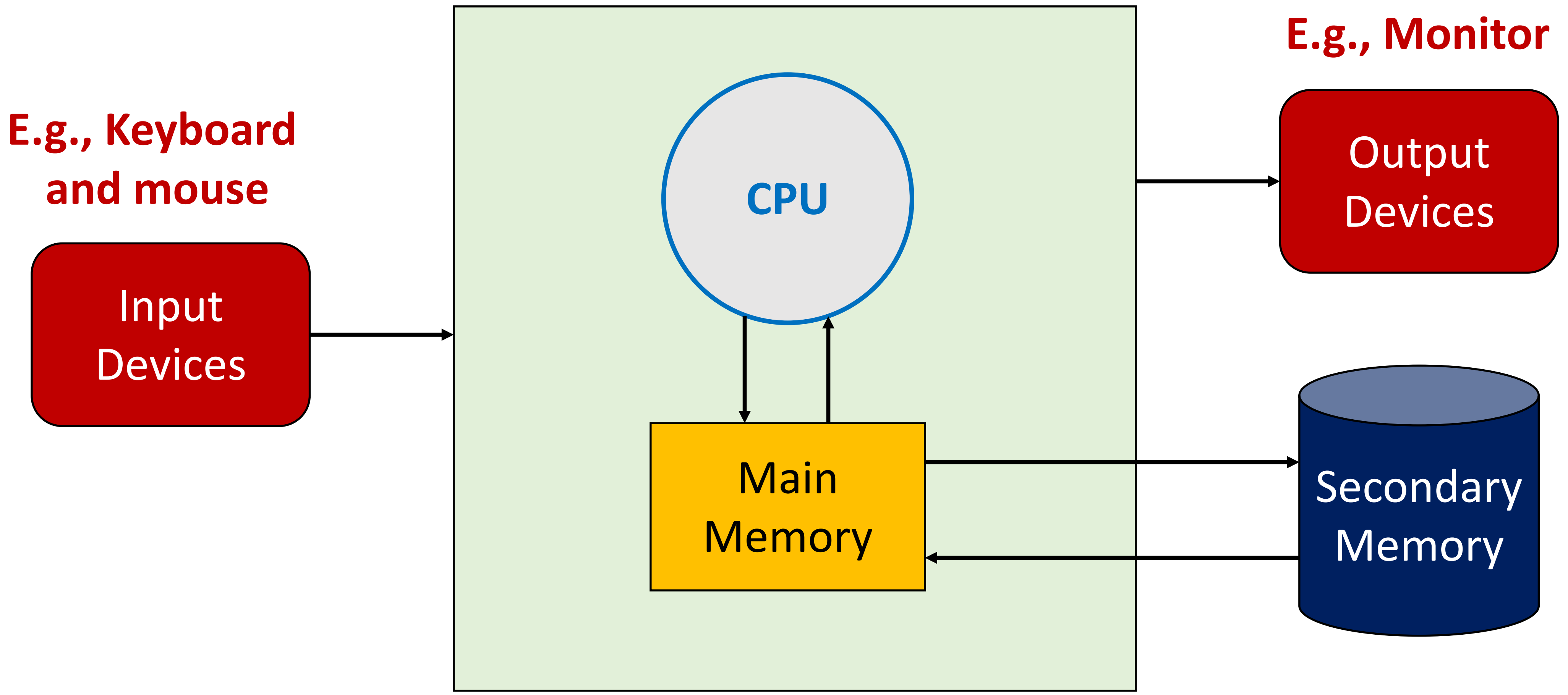
- The *Central Processing Unit* (CPU) is the "brain" of the computer

- It can at least perform:
  - Arithmetic operations (e.g., adding 2 numbers)
  - Logical operations (e.g., test if 2 numbers are equal)



- It can *directly* access information stored in main memory but not in secondary memory

# Functional View of a Computer



# Functional View of a Computer

**E.g., Keyboard and mouse**

Input Devices

- Humans interact with computers via Input and Output (IO) devices
- Information from Input devices are processed by the CPU and may be shuffled off to the main or secondary memory
- When information need to be displayed, the CPU sends them to one or more Output devices

**E.g., Monitor**

Output Devices

# Programming Languages

- A *program* is just a sequence of instructions telling the computer what to do
- Obviously, we need to provide these instructions in a language that computers can understand
  - We refer to this kind of a language as a *programming language*
  - Python, Java, C and C++ are examples of programming languages
- Every structure in a programming language has an exact form (i.e., *syntax*) and a precise meaning (i.e., *semantic*)
- A programming language is really a set of tools that allow us to program at a much higher level than the 0's and 1's that exist at the lowest levels of the computer.

# Machine Languages

- Python, Java, C, and C++ are, indeed, examples of *high-level* languages

- Strictly speaking, computer hardware can only understand a very *low-level* language known as *machine language*

- If you want a computer to add two numbers, the instructions that the CPU will carry out might be something like this:

Load the number from memory location 2001 into the CPU
Load the number from memory location 2002 into the CPU
Add the two numbers in the CPU
Store the result into location 2003

A Lot of Work!

# High-Level to Low-Level Languages

- In a high-level language like Python, the addition of two numbers can be expressed more naturally:

c = a + b

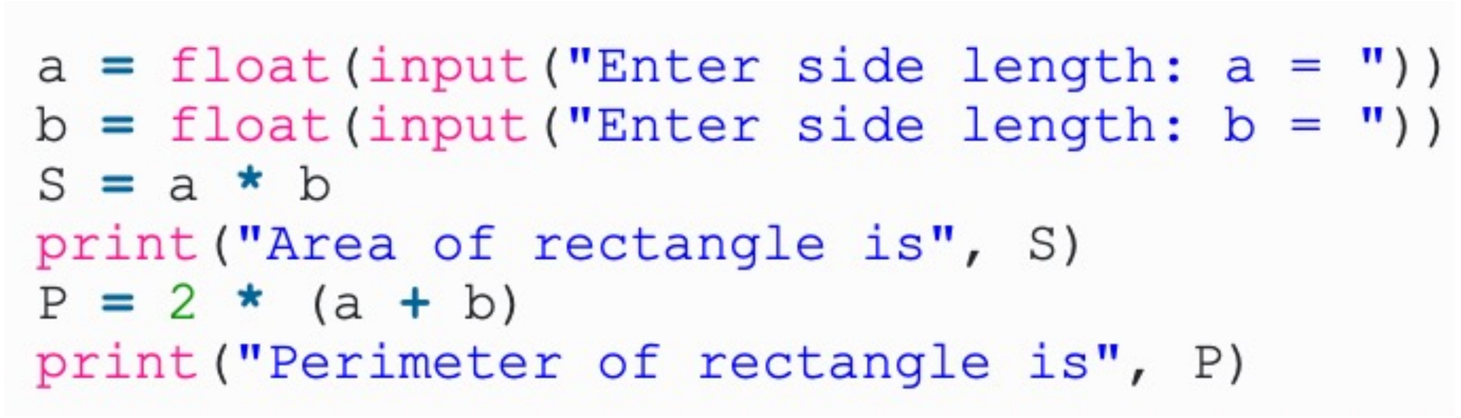
Much Easier!

- But, we need a way to translate the high-level language into a machine language that a computer can execute
  - To this end, high-level language can either be *compiled* or *interpreted*

## Examples: Rectangle

- Write a program that gets side lengths of a rectangle from the user and prints the area and perimeter of the rectangle:

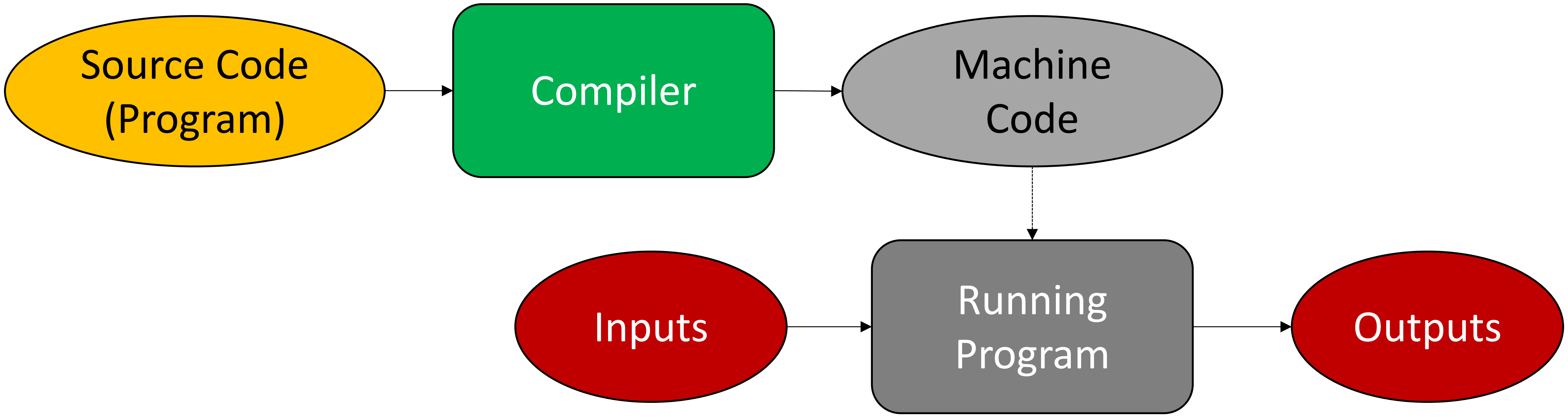
- The program gets the values of a and b from the user and converts them to floating-point numbers to be able to perform arithmetic operations with them.
- Then it computes the area, assigns the result to a third variable, and prints out the value of that variable. The same is repeated with the perimeter.



```python
a = float(input("Enter side length: a = "))
b = float(input("Enter side length: b = "))
S = a * b
print("Area of rectangle is", S)
P = 2 * (a + b)
print("Perimeter of rectangle is", P)
```

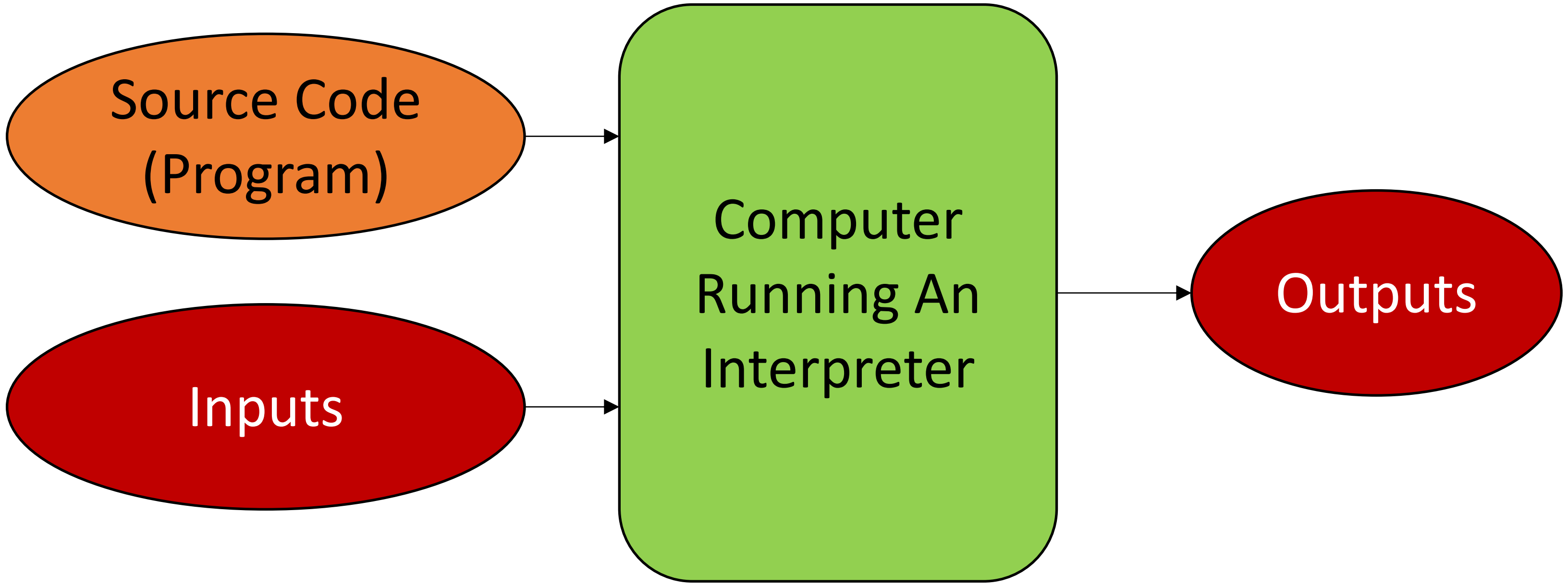
# Compiling a High-Level Language

- A *compiler* is a complex software that takes a program written in a high-level language and *translates* it into an equivalent program in the machine language of some computer



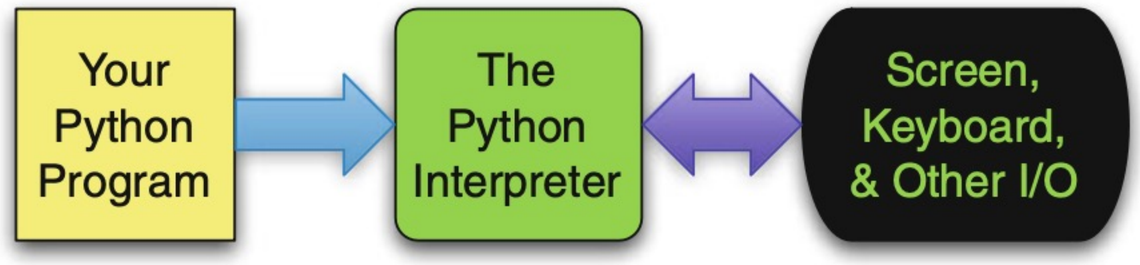
# Interpreting a High-Level Language

- An *interpreter* is a software that analyzes and executes the source code instruction-by-instruction (*on-the-fly*) as necessary



- E.g., Python is an interpreted language - To run a Python program you need an interpreter.

# The Python Interpreter



The interpreter is a program that interprets the programs we write.

## Compiling vs. Interpreting

- Compiling is a static (i.e., pre-execution), one-shot translation
  - Once a program is compiled, it may be run over and over again without further need for the compiler or the source code

- Interpreting is dynamic (i.e., happens during execution)
  - The interpreter and the source code are needed every time the program runs

- Compiled programs tend to be faster, while interpreted ones lend themselves to a more flexible programming environments (*they can be developed and run interactively*)

## Note on Portability

- The translation process highlights another advantage that high-level languages have over machine languages, namely, *portability*

- A program for an Intel-based machine will not run on an IBM-based machine since each computer type has its own machine language

- On the other hand, a program written in a high-level language (say, a Python program) can be run on many different kinds of computers as long as there is a suitable compiler or interpreter
    - Python programs are said to be portable!

## Recap

- A computer is a universal information-processing machine, which can carry out any process that can be described in sufficient detail
- A description of the sequence of steps for solving a particular problem is called an *algorithm*
- Algorithms can be turned into software (programs) that determine what the hardware (physical machine) can and does accomplish
- The process of creating software is called *programming*

## Recap

- A basic functional view of a computer system comprises a central processing unit (CPU), a main memory, a secondary memory, and input and output devices
- The CPU is the brain of the computer that performs simple arithmetic and logical operations
- Information that the CPU acts on (data and programs) are stored in main memory (e.g., RAM), while more permanent information are stored in secondary memory (e.g., disk)

## Recap

- Programs are written using a formal notation known as a *programming language*
- There are many different languages, but all share the property of having a precise *syntax* (form) and *semantics* (meaning)
- Computer hardware only understands a very low-level language known as *machine language*
- Programs are usually written using human-oriented *high-level languages* such as Python

## Recap

- A high-level language must either be *compiled* or *interpreted* in order for the computer to understand it
- High-level languages are more *portable* than machine languages
- Python is an interpreted language
- One way to learn about Python is to use an interactive shell for experimentation

## Recap

The following properties are required for developing a good program.

- **Program Correctness**: a good program must be able to solve the intended problem with relevant results.
- **Documentation**: every module or procedure must be preceded with comments on brief explanation of the module in the program.
- **Robustness and Scalability**: programs that can survive various unexpected events are said to be robust and those that can easily be upgraded are scalable.
- **User Interface**: a good look or design of the medium of interaction of the user and the program must be well taken care so as to have a good user interface, it must be easy and friendly to use.
- **Program Style**: the programming language rules are to be adapted in writing programs.
- **Use of Tools and Library Functions**: programming languages have special tools and libraries that can assist in developing computer programs. Some editors are also used to enhance the development of programs.

## Problem Solving

- Can you think of a day in your life which goes without problem solving?
- Whatever activity a human being or machine do for achieving a specified objective comes under problem solving.
- Problem is a kind of barrier to achieve something and *problem solving is a process to get that barrier removed by performing some sequence of activities*.
- If you can solve a given problem then you can also write an algorithm for it.
- If you can write an algorithm, then it is NOT difficult to describe a flowchart.

# Some example

- **Example1:** If you are watching a news channel on your TV and you want to change it to a sports channel, you need to do something i.e. *move to that channel by pressing that channel number on your remote.* **This is a kind of problem solving**

- **Example 2:** One Monday morning, a student is ready to go to school but yet he/she has not picked up those books and copies which are required as per timetable. So here *picking up books and copies as per timetable* is a **kind of problem solving** .

- **Example 3:** If someone asks you, what is the time now? In this case, *seeing time in your watch and telling him* is also a **kind of problem solving**.

- **Example 4:** Some students in a class plan to go on picnic and decide to share the expenses among them. In this case, *calculating total expenses and the amount an individual have to give for picnic* is also a **kind of problem solving.**

## Algorithm

- "A sequence of activities to be processed for getting desired output from a given input."

- "A formula or set of steps for solving a particular problem. To be an algorithm, *a set of rules must be unambiguous and have a clear stopping point*".

- **Note!** It is possible to have more than one way to solve a problem. Thus, there may be more than one algorithm for a problem.

## Algorithm: Remember this

1. Getting specified output is essential after algorithm is executed.

2. One will get output only if algorithm stops after finite time.

3. Activities in an algorithm must be clearly defined to avoid ambiguities.

## Algorithm: Remember this

- Before writing an algorithm for a problem, find out what is/are the inputs to the algorithm and what is/are expected output after running the algorithm.

- Symbols for different operations can be used to write an algorithm.

| | |
|---|---|
| ‘+’ | for Addition |
| ‘-’ | for Subtraction |
| ‘*’ | for Multiplication |
| ‘/’ | for Division and |
| ‘$\leftarrow$’ | for assignment. For example A $\leftarrow$ X*3 means A will have a value of X*3. |

## Algorithm: Example

- Problem 1: Find the area of a Circle of radius r.

  **Inputs to the algorithm:**

  Radius r of the Circle.

  **Expected output:**

  Area of the Circle

**Algorithm:**

```
Step1: Read\input the Radius r of the Circle
Step2: Area ⟵ PI*r*r // calculation of area
Step3: Print Area
```

# Algorithm: Example

- **Problem2:** Write an algorithm to read two numbers and find their sum.

  **Inputs to the algorithm:**

  First num1.

  Second num2.

  **Expected output:**

  Sum of the two numbers.

**Algorithm:**

```
Step1: Start
Step2: Read\input the first num1.
Step3: Read\input the second num2.
Step4: Sum ⟵ num1+num2 // calculation of sum
Step5: Print Sum
Step6: End
```

# Algorithm: Example

***Problem 3***: Convert temperature Fahrenheit to Celsius
Inputs to the algorithm:
Temperature in Fahrenheit
Expected output:
Temperature in Celsius

**Algorithm:**

```
Step1: Start
Step 2: Read Temperature in Fahrenheit F
Step 3: C ⟵ 5/9*(F32)
Step 4: Print Temperature in Celsius: C
Step5: End
```

## Type of Algorithm

1. Sequence.
2. Branching (Selection).
3. Loop (Repetition).

- These three control structures are sufficient for all purposes.
- Understanding them will help your programming skills – I promise 🫰

- The **sequence** is exemplified by sequence of statements place one after the other – the one above or before another gets executed first.
- In flowcharts, **sequence of statements** is usually contained in the rectangular process box. *I will show this in later slide.*

# Type of Algorithm

1. Sequence.

2. Branching (Selection).

3. Loop (Repetition).

- The **branch** refers to a binary decision based on some condition.
- If the condition is **true**, one of the two **branches is explored**; if the condition is **false**, the **other alternative is taken**.
- Branches are  usually represented by the **'if-then'** construct in pseudo-codes and programs.
- In flowcharts, this is represented by the diamond-shaped decision box. ***I will show this in later slide***.
- The **branch** structure is also known as the **selection structure.**

## Example of Branching Algorithm

***Problem1***: write algorithm to find the greater number between two numbers

```
Step1: Start
Step2: Read/input A and B
Step3: If A greater than B then C=A
Step4: if B greater than A then C=B
Step5: Print C
Step6: End
```

## Example of Branching Algorithm…

***Problem2***: write algorithm to find the result of equation: $f(x) = \begin{cases} -x, & x < 0 \\ x, & x \geq 0 \end{cases}$

Step1: Start

Step2: Read/input x

Step3: If X Less than  zero  then F=-X

Step4: if  X greater than or equal zero then F=X

Step5: Print F

Step6: End

## Example of Branching Algorithm…

***Problem3***: A algorithm to find the largest value of any three numbers.

```
Step1: Start

Step2: Read/input A,B and C

Step3: If (A>=B) and (A>=C) then Max=A

Step4: If (B>=A) and (B>=C) then Max=B

Step5:If (C>=A) and (C>=B) then Max=C

Step6: Print Max

Step7: End
```

## Type of Algorithm

1. Sequence.
2. Branching (Selection).
3. Loop (Repetition).

- ✓ Note: Ensure that the condition for the termination of the looping must be satisfied after some finite number of iterations,
  - ✓ otherwise it ends up as an infinite loop,
    - ✓ This is a common mistake made by inexperienced programmers.
- ✓ The loop is also known as the repetition structure

- The **loop** allows a statement or a sequence of statements to be repeatedly executed based on some **loop condition**.
- It is represented by the **'while' and 'for'** constructs in most programming languages, for unbounded loops and bounded loops respectively.
  - (Unbounded loops refer to those whose number of iterations depends on the eventuality that the termination condition is satisfied;
  - bounded loops refer to those whose number of iterations is known before-hand.).
- In the flowcharts, a back arrow hints the presence of a loop. ***I will show this in later slide***
- A trip around the loop is known as iteration.

## Example of Looping Algorithm

***Problem1***: An algorithm to calculate even numbers between 0 and 99

```
1. Start
2. I ← 0
3. Write I in standard output
4. I ← I+2
5. If (I <=98) then go to line 3
6. End
```

## Example of Looping Algorithm…

***Problem2***: Design an algorithm which gets a natural value, n,as its input and calculates odd numbers equal or less than n. Then write them in the standard output:

```
1. Start
2. Read n
3. I ← 1
4. Write I
5. I ← I + 2
6. If ( I <= n) then go to line 4
7. End
```

## Example of Looping Algorithm…

***Problem3***: Design an algorithm which generates even numbers between 1000 and 2000 and then prints them in the standard output. It should also print total sum:

```
1. Start
2. I ← 1000 and S ← 0
3. Write I
4. S ← S + I
5. I ← I + 2
6. If (I <= 2000) then go to line 3
else go to line 7
7. Write S
8. End
```

## Example of Looping Algorithm…

***Problem4***: Design an algorithm with a natural number, n, as its input which calculates the following formula and writes the result in the standard output:

$S = \frac{1}{2} + \frac{1}{4} + \dots + 1/n$

```
1. Start
2. Read n
3. I ← 2 and S ← 0
4. S= S + 1/I
5. I ← I + 2
6. If (I <= n) then go to line 4
   else write S in standard output
7. End
```

## Let’s Explore Python Programming Language (PL)

## Python!

- Created in 1991 by Guido van Rossum (now at Google)
  - Named for Monty Python

- Useful as a **scripting language**
  - **script**: A small program meant for one-time use
  - Targeted towards small to medium sized projects

- Used by:
  - Google, Yahoo!, Youtube
  - Many Linux distributions
  - Games and apps (e.g. Eve Online)

## Things to Know about Python Programming Language (PL)

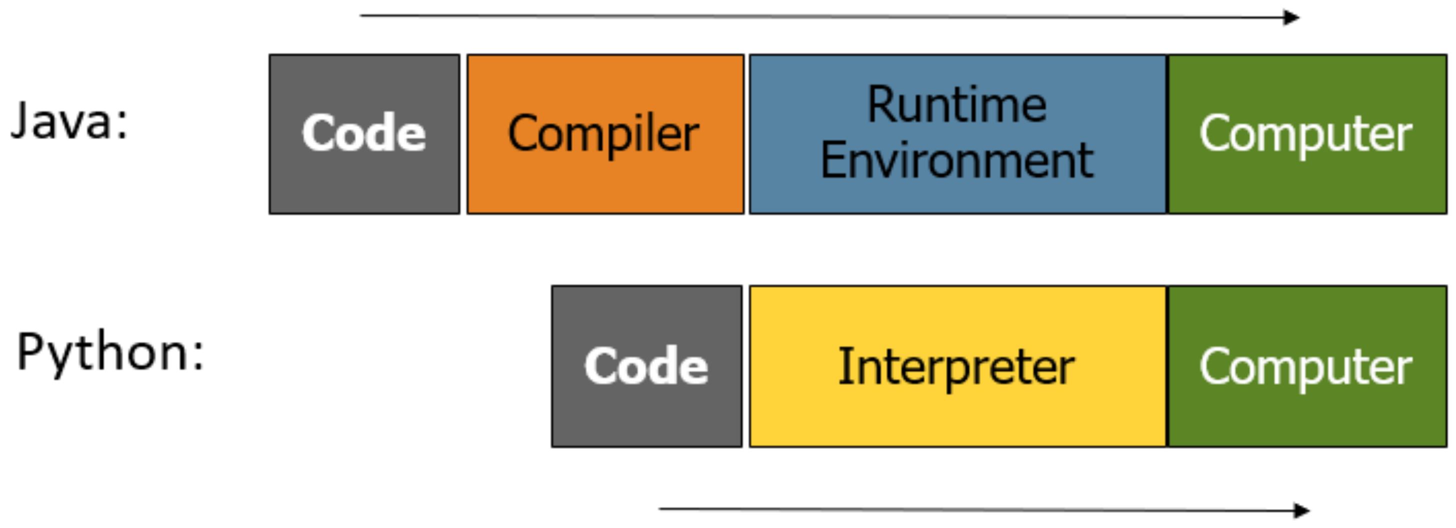
- Python is a well-known high-level programming language that was first introduced in 1991.
- Python is a high-level, interpreted, interactive and object oriented-scripting language.
- Python is mostly used for **server-side web development**, **software development**, **math, scripting, and artificial intelligence (AI).**
- Python runs on a variety of systems, including Windows, Mac OS X, Linux, and Raspberry Pi.
- Python PL is Open source and community driven
- Python was designed to be highly readable which uses English keywords frequently where as other languages use punctuation and it has fewer syntactical constructions than other languages.
- Python has several Integrated Development Environment (IDE)

## What is Unique About Python?

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- High-level internal object data types
- Automatic memory management
- It’s free (open source)!

# Interpreted Languages

- **interpreted**
  - Not compiled like Java
  - Code is written and then directly executed by an **interpreter**
  - Type commands into interpreter and see immediate results



## Language (Python) properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure

## Why learn python?

- Fun-to-use "Scripting language"
- Object-oriented
    - Highly educational
- Very easy to learn
- Powerful, scalable, easy to maintain
    - high productivity
    - Lots of libraries
- Glue language
    - Interactive front-end for FORTRAN/C/C++ code

## Why learn python? (cont.)

- Reduce development time
- Reduce code length
- Easy to learn and use as developers
- Easy to understand codes
- Easy to do team projects
- Easy to extend to other languages

## Where to use python?

- System management (i.e., scripting)
- Graphic User Interface (GUI)
- Internet programming
- Database (DB) programming
- Text data processing
- Distributed processing
- Numerical operations
- Graphics
- And so on…

## Python IDE

- IDE stands for Integrated Development Environment
- It is a software package that includes tools for designing and testing software.
- Throughout the SDLC, a developer employs a variety of tools like as editors, libraries, compilers, and testing platforms.
- The IDE aids in the automation of a developer's process by eliminating manual efforts and bringing all of the tools together in a single framework.
- If the IDE isn't installed, the developer will have to conduct the selections, integrations, and deployment process manually.
- The IDE was created to make the SDLC process easier by decreasing coding and preventing typing errors.
- In contrast to the IDE, some developers also prefer Code editors.

## Python IDE

- A **code editor** is essentially a **text editor** in which a programmer can create code for the development of any software.
    - A tool that is used to write and edit code.

- The developer can also save small text files for the code in the code editor.

- In comparison to IDE, code editors are fast in operating and have a small size.

- Code editors possess the capability of executing and debugging code.

## Python IDE

- IDE understand your code much better than a text editor.
- It usually provides features such as build automation, code linting, testing and debugging.
- IDEs significantly speed up your work.
- The downside is that IDEs can be complicated to use.
  - But don’t be discourage! Just use it and keep smiling!!

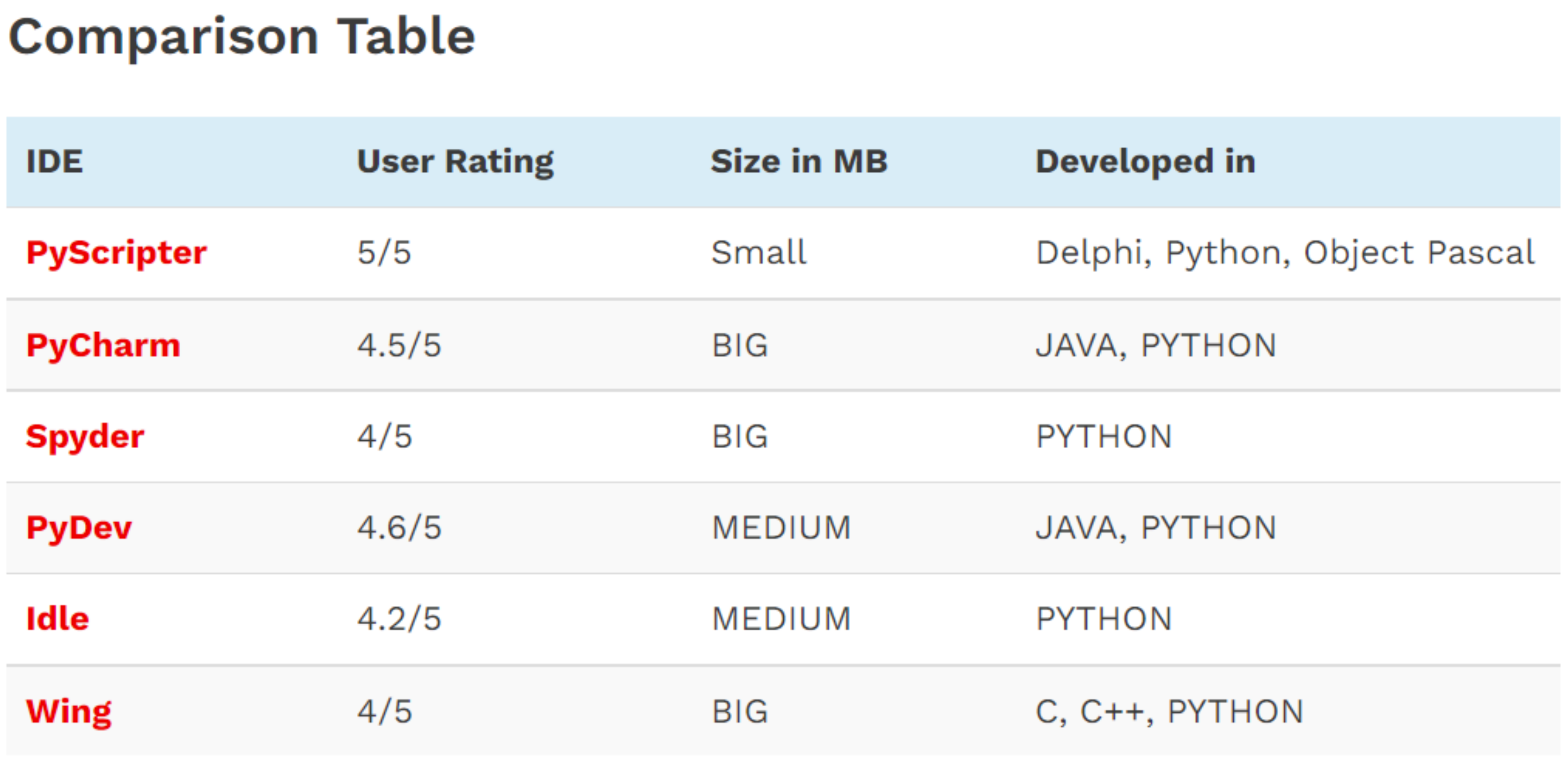
Read more here:
https://www.softwaretestinghelp.com/python-ide-code-editors/
https://www.guru99.com/python-ide-code-editor.html
https://realpython.com/python-ides-code-editors-guide/
https://hackr.io/blog/best-python-ide
https://www.programiz.com/python-programming/ide

**Python IDE**

i. PyScripter
ii. PyCharm
iii. Spyder
iv. Pydev
v. Idle
vi. Wing
vii. Eric Python
viii. Thonny
ix. Rodeo

**Python Code Editor**

i. Sublime Text
ii. Atom
iii. Vim
iv. Visual Studio Code



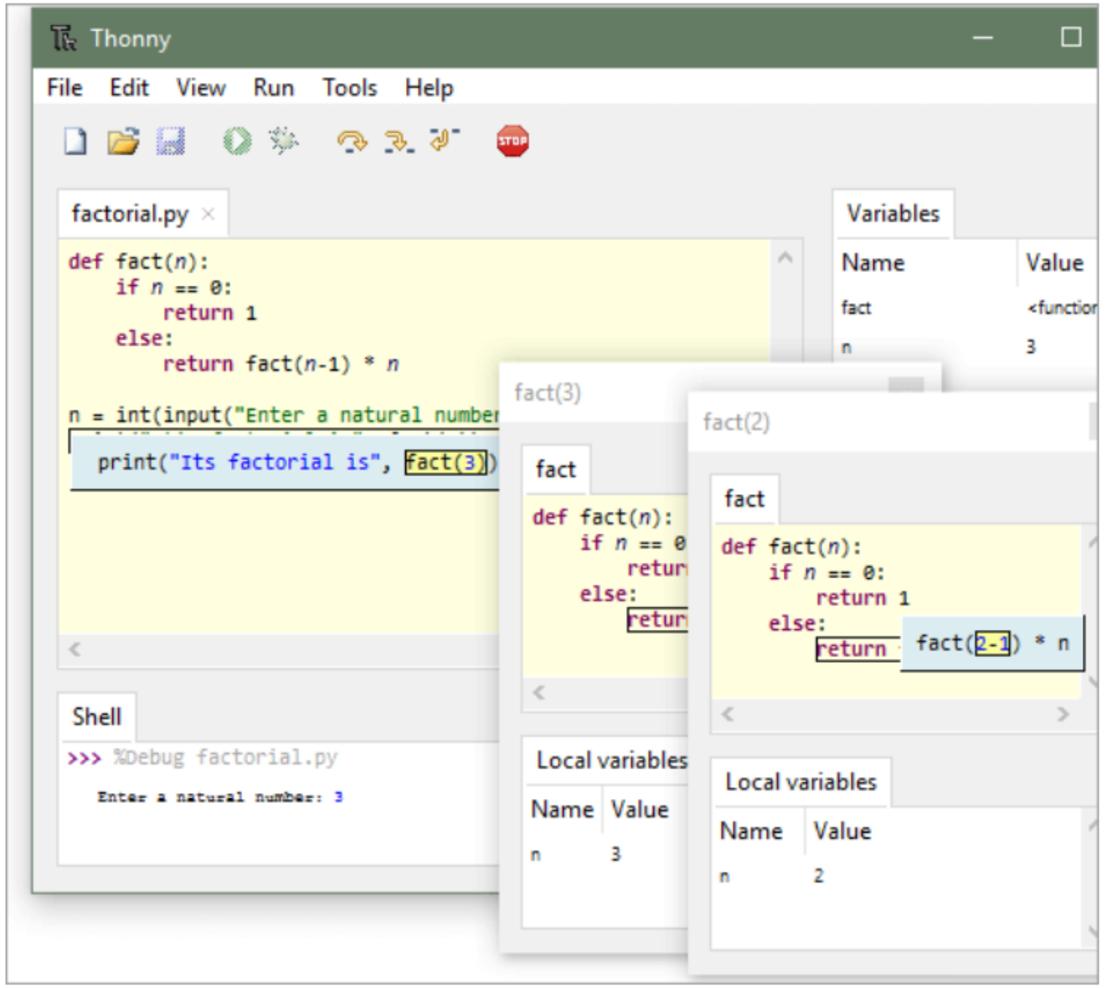
## Comparison Table

| IDE | User Rating | Size in MB | Developed in |
|---|---|---|---|
| **PyScripter** | 5/5 | Small | Delphi, Python, Object Pascal |
| **PyCharm** | 4.5/5 | BIG | JAVA, PYTHON |
| **Spyder** | 4/5 | BIG | PYTHON |
| **PyDev** | 4.6/5 | MEDIUM | JAVA, PYTHON |
| **Idle** | 4.2/5 | MEDIUM | PYTHON |
| **Wing** | 4/5 | BIG | C, C++, PYTHON |

# The IDE for CSC201

**Thonny**

- ✓ Thonny is a Python dedicated IDE that comes with Python 3 built-in.
- ✓ Once you install it, you can start writing Python code.
- ✓ Thonny is intended for beginners.
- ✓ The user interface is kept simple so that beginners will find it easy to get started.
- ✓ Thonny is intended for beginners, it has several useful features that also make it a good IDE for full-fledged Python development.
- ✓ Some of its features are syntax error highlighting, debugger, code completion, step through expression evaluation, etc.



```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(n-1) * n

n = int(input("Enter a natural number
print("Its factorial is", fact(3))
```

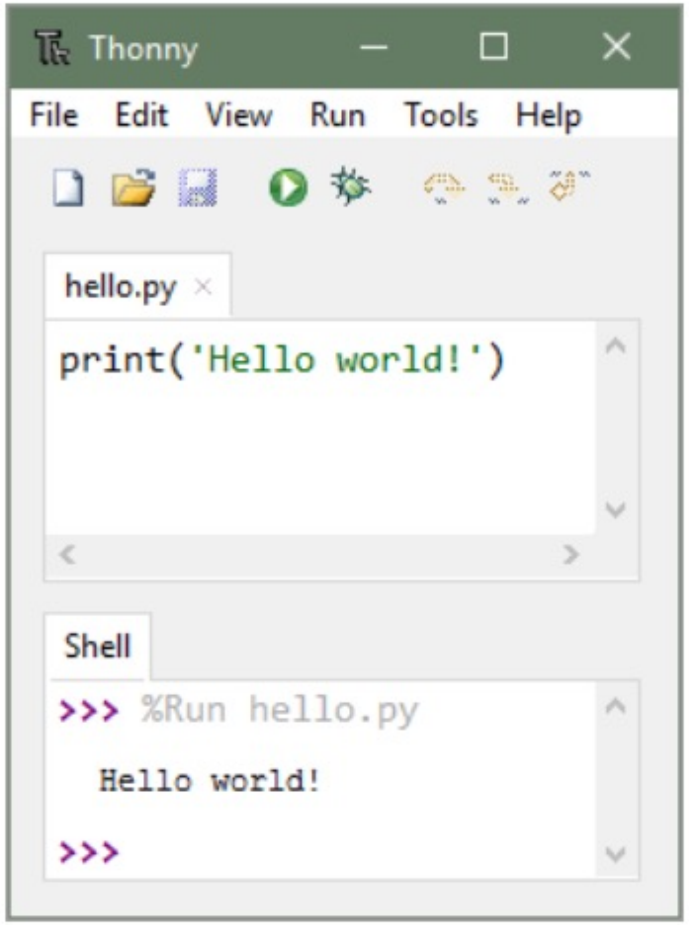
```
>>> %Debug factorial.py
 Enter a natural number: 3
```

| Name | Value |
|---|---|
| fact | <function |
| n | 3 |

```
def fact(n):
    if n == 0:
        return 1
    else:
        return fact(2-1) * n
```

| Name | Value |
|---|---|
| n | 3 |

| Name | Value |
|---|---|
| n | 2 |



```
print('Hello world!')
```

```
>>> %Run hello.py
  Hello world!
>>>
```

## The IDE for CSC201 Thonny

## Watch this 👇

https://www.youtube.com/watch?v=nwIgxrXP-X4 (Introducing Thonny)
https://www.youtube.com/watch?v=Y8Tko2YC5hA (What and why Python)
https://www.youtube.com/watch?v=1GqMpz77XlU (Time taken to learn Python)
https://www.youtube.com/watch?v=Z1Yd7upQsXY (On Variables)
https://www.youtube.com/watch?v=AWek49wXGzI (If Else Statements)
https://www.youtube.com/watch?v=NSbOtYzIQI0 (Functions In Python)
https://www.youtube.com/watch?v=tw7ror9x32s (Lists In Python)
https://www.youtube.com/watch?v=OnDr4J2UXSA (For Loops in Python)
https://www.youtube.com/watch?v=mElVGah7Epg (Motivation on learning code)
https://www.youtube.com/watch?v=poJfwre2PIs (Advise on PL to Learn First?)
https://www.youtube.com/watch?v=rfscVS0vtbw (Full introduction on python)

# Python Syntax

- Much of it is similar to C syntax
- Exceptions:
  - missing operators: `++`, `--`
  - no curly brackets, `{ } ,` for blocks; uses whitespace
  - different keywords
  - lots of extra features
  - no type declarations!

## Try this Yourself!

- Download and Install the Thonny IDE
- Download Python Interpreter.
- Install the Python Programming Language

## Home work

- ✓ Read and understand the **features**, **pros** and **cons** of the Python IDEs in https://www.softwaretestinghelp.com/python-ide-code-editors/ and respond to the following:

  a. Write down your reflections and impressions on these IDEs.
  b. Why would you prefer to use IDEs instead of code editors or vis-versa
  c. Which is the best IDE for you and why?
  d. Do you think Python Programming is Difficult?
  e. Did you find it difficult to Download and Install Thonny IDE?

**Submit your response to through this link:** https://forms.gle/HPUZiu7sPcfxPXLv5

# Installing Python

**Windows:**

- Download Python from http://www.python.org
- Install Python.
- Run **Idle** from the Start Menu.

**Note:** For step by step installation instructions, see the course web site.

**Mac OS X:**

- Python is already installed.
- Open a terminal and run `python` or run Idle from Finder.

**Linux:**

- Chances are you already have Python installed. To check, run `python` from the terminal.
- If not, install from your distribution's package system.

## Note the following

- Binaries of latest version of Python 3 are available on Python Official Website.
- For versions 3.0 to 3.4.x, Windows XP is acceptable.
- In order to install Python 3.5 to 3.8- minimum OS requirements are Windows 7 with SP1.
- Python 3.9+ cannot be used on Windows 7 or earlier.
- Python 3.10.0 is the newest major release of the Python programming language
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.
- You can download Python documentation from the site. The documentation is available in HTML, PDF and PostScript formats.
  - ✓ Python Official Website: http://www.python.org/
- You can create and run python codes on your mobile devices.
  - ✓ There are so many python 3 interpreter and IDE available on Google and IOS play store. For example, Pydroid 3 on Android phone.

## Features of Python

- **Python is Interpreted**: This means that it is processed at runtime by the interpreter and you do not need to compile your program before executing it. This is similar to PERL and PHP.

- **Python is Interactive**: This means that you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- **Python is Object-Oriented**: This means that Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is Beginner's Language**: Python is a great language for the beginner programmers and supports the development of a wide range of applications, from simple text processing to WWW browsers to games.

## Features of Python…

- **Easy-to-learn:** Python has relatively **few keywords**, **simple structure**, and a clearly defined syntax.
  - So it is possible to consider learning the language in a relatively short period of time.
- **Easy-to-read**: Python code is much more clearly defined and visible to the eyes.
- **Easy-to-maintain**: Python's success is that its source code is fairly easy-to-maintain.
- **A broad standard library**: One of Python's greatest strengths is the bulk of the library. It is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode**: You can enter results from a terminal right to the language, allowing interactive testing and debugging of snippets of code.

## Features of Python…

- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter.
  - These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

**Microsoft Foundation Class (MFC)**

## Additional Features of Python

- Support for functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- Very high-level dynamic data types and supports dynamic type checking.
- Supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

# Writing Python Commands

- You can start the Python interpreter in an *interactive mode* and type in some commands to see what happens

- When you first start the interpreter program, you may see something like the following:

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 19 2015, 20:38:52)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

A Python *prompt* indicating that the Python interpreter is waiting for you to give it a command

# Writing Python Commands

- Here is a sample interaction with the Python interpreter:

```
>>> print("Hello")
Hello
>>> print("Programming is fun!")
Programming is fun!
>>> print(3)
3
>>> print(2.3)
2.3
```

- *print* is a built-in function that allows displaying information on screen
- When you call the print function, the *parameters* in the parentheses tell the function what to print
- There is only 1 parameter passed to print here, which is a *textual data* (or what is referred to as a ***string***)

# Writing Python Commands

- Here is a sample interaction with the Python interpreter:

```
>>> print("Hello")
Hello
>>> print("Programming is fun!")
Programming is fun!
>>> print(3)
3
>>> print(2.3)
2.3
```

Another string parameter

An *integer* (*int* for short) parameter

A *float* parameter

How can we figure out in a program whether a value is an int, float, or string?

# The Type Function

- Python provides a special function called *type*, which allows us to figure out the *data type* of any value

```
>>> type("Hello")
<class 'str'>
>>> type("Programming is fun!")
<class 'str'>
>>> type(3)
<class 'int'>
>>> type(2.3)
<class 'float'>
```

## First Python Program

•**Interactive Mode Programming**
Invoking the interpreter without passing a script file as a parameter brings up the following prompt-

```
$ python
Python 3.3.2 (default, Dec 10 2013, 11:35:01)
[GCC 4.6.3] on Linux
Type "help", "copyright", "credits", or "license" for more information.
>>>
On Windows:
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Type the following text at the Python prompt and press Enter-

```
>>> print ("Hello, Python!")
```

## Program Elements

i. Identifiers

ii. Variables

iii. Reserved words

# Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.

- An identifier starts with a letter **A to Z** or **a to z** or an underscore ( _ ) followed by zero or more letters, underscores and **digits (0 to 9)**.

- All identifiers must begin with letter or underscore, followed by any number of letters, digits, underscores.

## Rules for writing identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _
- Names like myClass, var_1 and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
- Keywords cannot be used as identifiers.

```
>>> global = 1
  File "<interactive input>", line 1
    global = 1
           ^
SyntaxError: invalid syntax
```

## Rules for writing identifiers

- Do not use special symbols like !, @, #, $, % etc., your identifier.

```
>>> a@ = 0
  File "<interactive input>", line 1
    a@ = 0
     ^
SyntaxError: invalid syntax
```

- Identifier can be of any length.

## What more?

- Python is a case-sensitive language.
  - This means, Variable and variable are not the same.
  - Also, **Manpower** and **manpower** are two different identifiers in Python.
  - Always name identifiers that make sense.

## Variables

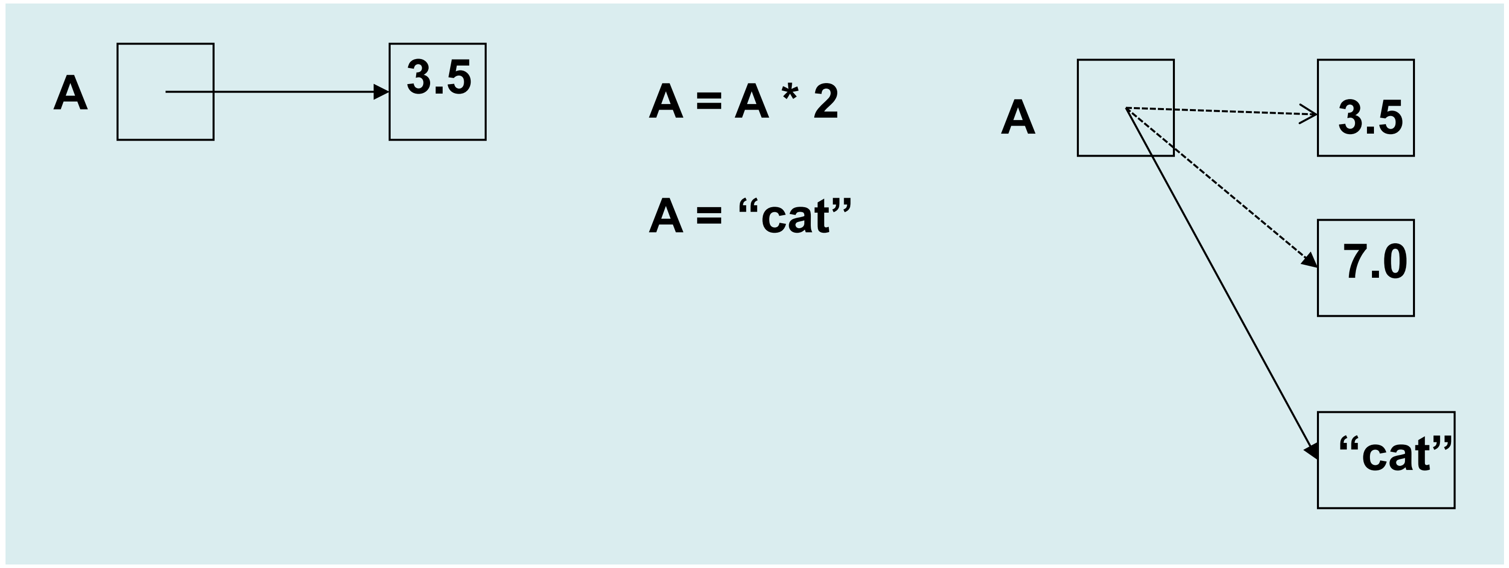
- Variables:
  - Do not need to be declared
  - A variable is created when a value is assigned to it: Examples: num = 3
  - Can’t be used in an expression unless it has a value
  - Error message: *Name Error* – means no value is associated with this name

## Variables

- Variable names don’t have static types – values (or objects) do
  - A variable name has the type of the value it currently references

- Variables actually contain references to values (similar to pointers).
  - This makes it possible to assign different object types to the same variable

- Variables contain references to data values



- Python handles memory management automatically.
  - It creates new objects and stores them in memory;
  - It will also execute garbage collection algorithms to reclaim any inaccessible memory locations.
- Python does not implement reference semantics for simple variables;
  - For example: if A = 10 and B = A, A = A + 1 does not change the value of B

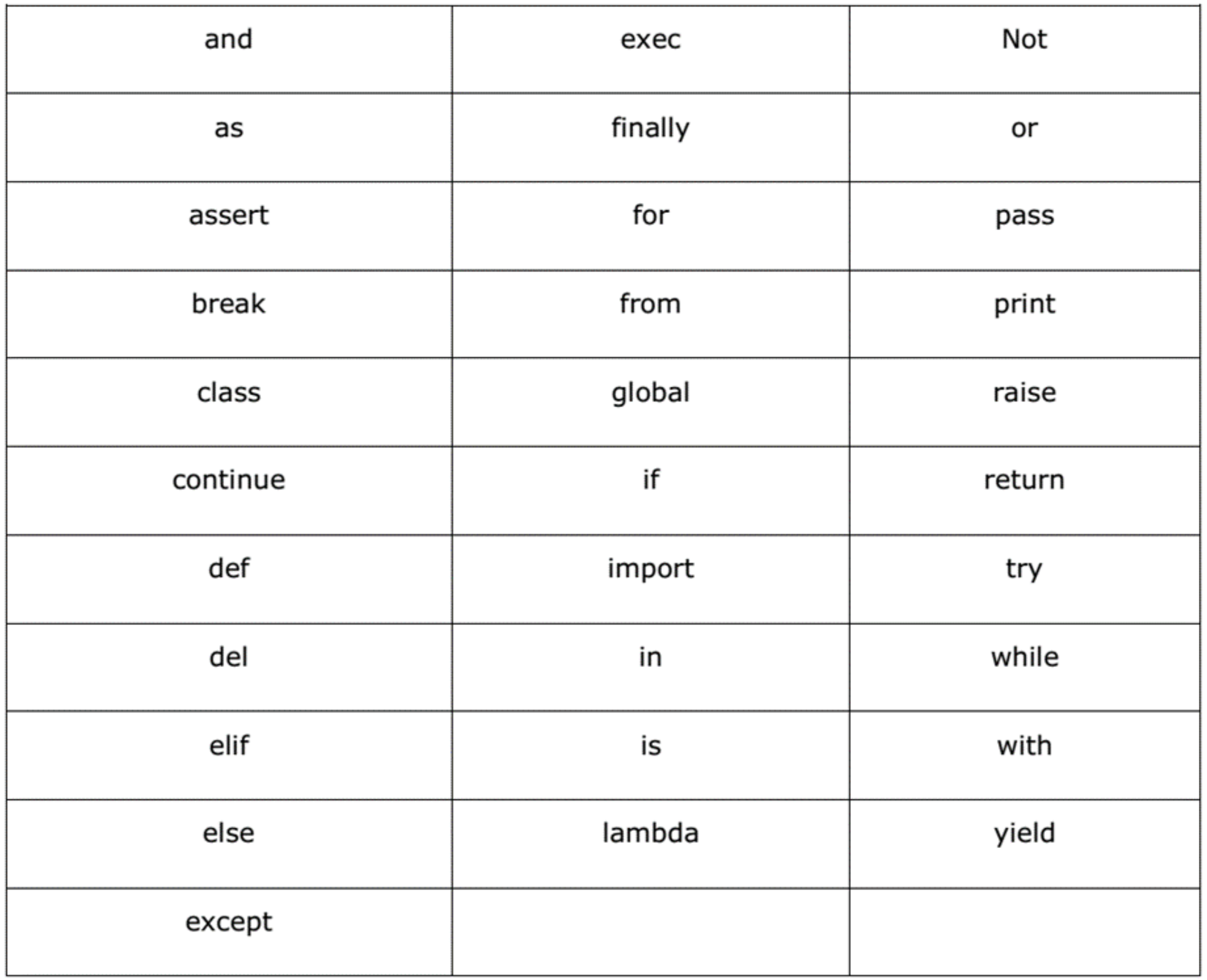
# More on Variables

- In many programming languages, variables are statically typed.
    - That means a variable is initially declared to have a specific data type, and any value assigned to it during its lifetime must always have that type.
- Variables in Python are not subject to this restriction.
- In Python, a variable may be assigned a value of one type and then later re-assigned a value of a different type:

```
>>> var = 23.5
>>> print(var)
23.5

>>> var = "Now I'm a string"
>>> print(var)
Now I'm a string
```

# Reserved Words

- Reserved words are also refer to **keywords**.
- Reserved words are **NOT** use as constants or variables or any other identifier names.
- All the Python reserved words/keywords contain **lowercase letters** only.



| | | |
|---|---|---|
| and | exec | Not |
| as | finally | or |
| assert | for | pass |
| break | from | print |
| class | global | raise |
| continue | if | return |
| def | import | try |
| del | in | while |
| elif | is | with |
| else | lambda | yield |
| except | | |

## Lines and Indentation

- Python does not use braces ({}) to indicate blocks of code for class and function definitions or flow control.
  - Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example-

```
if True:
    print ("True")
else:
  print ("False")
```

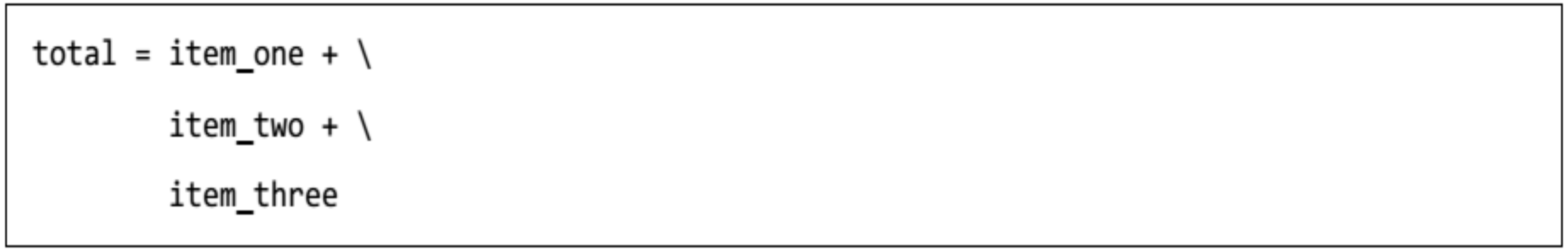
However, the following block generates an error-

```
if True:
    print ("Answer")
    print ("True")
else:
    print "(Answer")
  print ("False")
```

- In Python all the continuous lines indented with the same number of spaces would form a block.

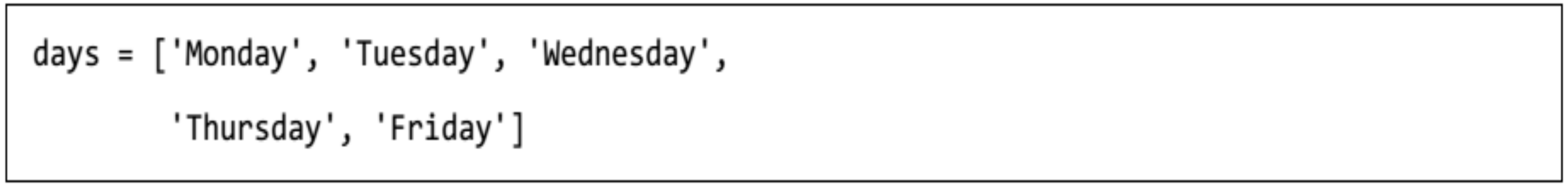
## Multi-Line Statements

- Statements in Python typically end with a new line.
- Python, however, allows the use of the line continuation character (\) to denote that the line should continue. For example



```
total = item_one + \
        item_two + \
        item_three
```

- The statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example

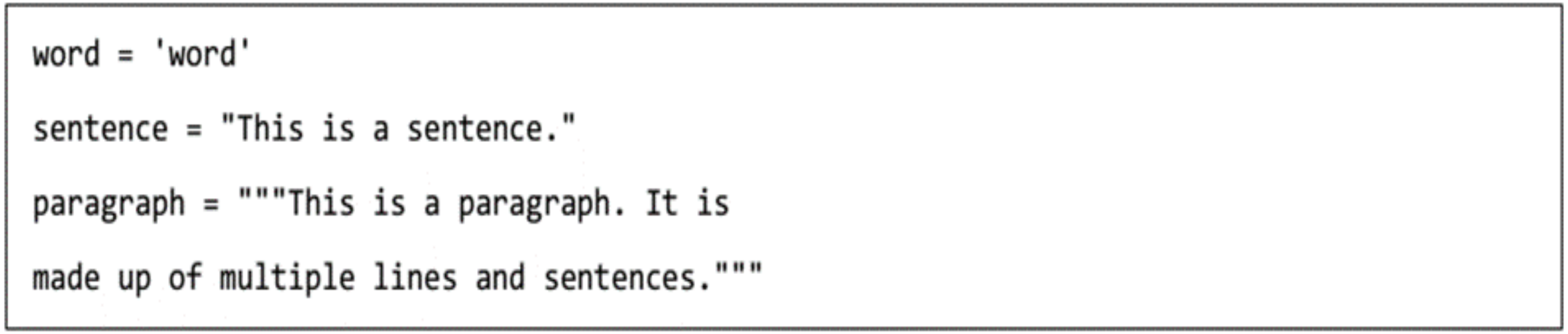


```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

## Quotation in Python

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

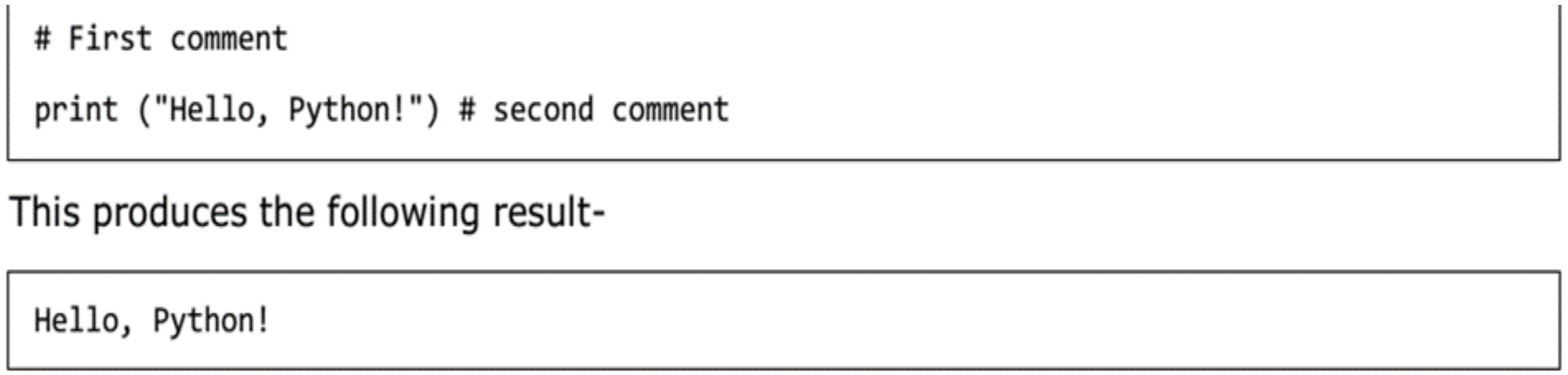
- The triple quotes are used to span the string across multiple lines.
  – For example, all the following are legal



```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""
```

# Comments in Python

- A hash sign (#) that is not inside a string literal is the beginning of a comment.
  - All characters after the #, up to the end of the physical line, are part of the comment and the
  - Python interpreter ignores them (the comment).



```
# First comment

print ("Hello, Python!") # second comment
```

This produces the following result-

```
Hello, Python!
```

## More on Comment in Python

- You can type a comment on the same line after a statement or expression

```
name = "Madisetti" # This is again comment
```

- Python does not have multiple-line commenting feature.
  - You have to comment each line individually as follows:

```
# This is a comment.

# This is a comment, too.

# This is a comment, too.

# I said that already.
```

## Using Blank Lines

- A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

- In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

## Interesting Features

- White space does indicate meaning
  - Instead of curly brackets or begin-end pairs, whitespace is used for block delimiters.
  - Indent statements in a block, un-indent at end of block.
- Statements are terminated by <Enter>
- No variable declarations
- Dynamic typing
- Associative arrays (dictionaries)
- Lists and slices

## Additional Material

- https://www.stat.berkeley.edu/~spector/python.pdf

## Next Lecture

- Input Statements
- Output Statements
- Assignment Statements
- Operators and Operands
- Precedence Rule
- Expressions in Python
- Data types in Python