Contents

Preface

The creation of this book has been for me, Mike McGrath, an exciting personal journey in discovering how Python can be used today for procedural and object-oriented programming, to develop applications and to provide online functionality. Example code listed in this book describes how to produce Python programs **in easy steps** – and the screenshots illustrate the actual results. I sincerely hope you enjoy discovering the exciting possibilities of Python, and have as much fun with it as I did in writing this book.

In order to clarify the code listed in the steps given in each example I have adopted certain colorization conventions. Components of the Python programming language are colored blue, programmer-specified names are red, numeric and string data values are black, and comments are green, like this:

```
# Write the traditional greeting.
greeting = 'Hello World!'
print( greeting )
```

Additionally, in order to identify each source code file described in the steps, a colored icon and file name appears in the margin alongside the steps:

script.py

page.html

image.gif

For convenience I have placed source code files from the examples featured in this book into a single ZIP archive. You can obtain the complete archive by following these easy steps:

1. Browse to **www.ineasysteps.com** then navigate to Free Resources and choose the Downloads section

2. Find Python in easy steps, 2nd edition in the list, then click on the hyperlink entitled All Code Examples to download the archive

3. Next, extract the **MyScripts** and **MyProjects** folders to your home directory (such as **C:**) and copy all contents of the **htdocs** folder to your web server's documents directory

4. Now, follow the steps to call upon the Python interpreter and see the output

Book resources and updates are also available on the product page at **www.ineasysteps.com**

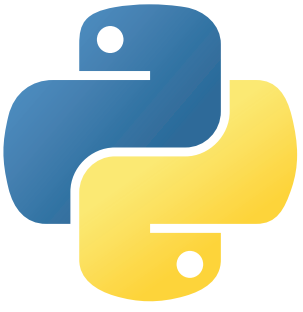
If you don't achieve the result illustrated in any example, simply compare your code to that in the original example files you have downloaded to discover where you went wrong.

1 Getting started

Welcome to the exciting world of the Python programming language. This chapter demonstrates how to install Python and create your first program.

Introducing Python

Python is a high-level (human-readable) programming language that is processed by the Python "interpreter" to produce results. Python includes a comprehensive standard library of tested code modules that can be easily incorporated into your own programs.



Discover all the latest Python news online at **www.python.org**

The Python language was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including C, C++, the Unix shell and other programming languages. Today, Python is maintained by a core development team at the Institute, although Guido van Rossum still holds a vital role in directing its progress.

The basic philosophy of the Python language is readability, which makes it particularly well-suited for beginners in computer programming, and it can be summarized by these principles:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Programming languages that group blocks by indentation are said to adhere to the "offside rule" – a pun on the offside rule in soccer.

As Python is intended to be highly readable, it uses English keywords frequently where other languages may use punctuation. Most significantly, it uses indentation to group together statements into code "blocks", whereas other languages may use keywords or punctuation for this purpose. For example, in the Pascal programming language, blocks start with the keyword **begin** and end with the keyword **end**, and in the C programming language, blocks are enclosed within curly brackets (**{ }** braces). Grouping blocks of statements by indentation is sometimes criticized by programmers familiar with languages that group by punctuation, but the use of indentation in Python certainly produces code that has an uncluttered visual layout.

Some of Python's key distinguishing features that make it an attractive choice of language for the beginner include:

- **Python is free** – is open source distributable software.
- **Python is easy to learn** – has a simple language syntax.
- **Python is easy to read** – is uncluttered by punctuation.
- **Python is easy to maintain** – is modular for simplicity.
- **Python is "batteries included"** – provides a large standard library for easy integration into your own programs.
- **Python is interactive** – has a terminal for debugging and testing snippets of code.
- **Python is portable** – runs on a wide variety of hardware platforms and has the same interface on all platforms.
- **Python is interpreted** – there is no compilation required.
- **Python is high-level** – has automatic memory management.
- **Python is extensible** – allows the addition of low-level modules to the interpreter for customization.
- **Python is versatile** – supports both procedure-orientated programming and object-orientated programming (OOP).
- **Python is flexible** – can create console programs, windowed GUI (Graphical User Interface) applications, and CGI (Common Gateway Interface) scripts to process web data.

Python is named after the British television comedy series "Monty Python's Flying Circus" – you may encounter references to this in the Python documentation.

As development of Python continues, newer versions are released as with most software. Python 2.7 version was the final release of the 2.x series, for which support ended on January 1, 2020.

The Python 3.x series has already seen several stable releases, and all recent standard library improvements are only available in Python 3.x. This book describes and demonstrates features of the present and the future of Python with the latest 3.x version.

Python 3.x is not backward compatible with Python 2.x series.

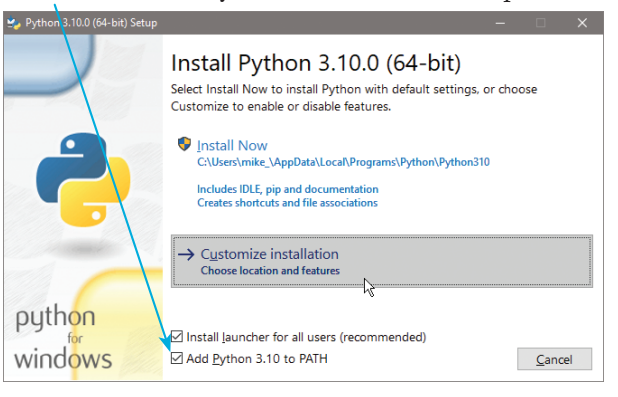
Installing Python on Windows

Before you can begin programming in the Python language you need to install the Python interpreter on your computer, and the standard library of tested code modules that comes along with it. This is available online as a free download from the Python website at **https://python.org/downloads**. For Windows users there are installers available in both 32-bit and 64-bit versions:

Installers for macOS/Mac OS X in 32-bit and 64-bit versions are available at **python.org/downloads**

1. Launch a web browser, then navigate to **python.org/downloads** and download the appropriate installer for your system – in this example it's a file named "python-3.10.0-amd64.exe"

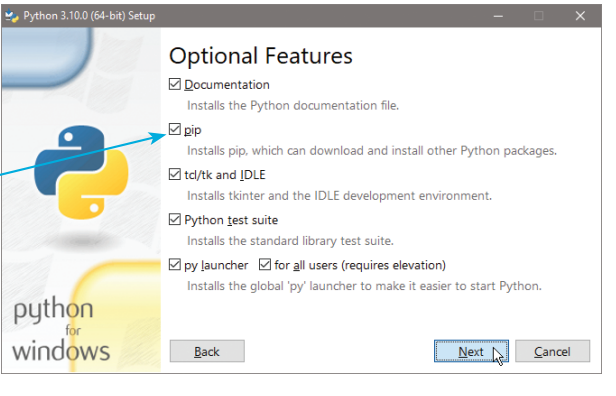
2. When the download completes, run the installer and check the "Add Python 3.10 to PATH" option



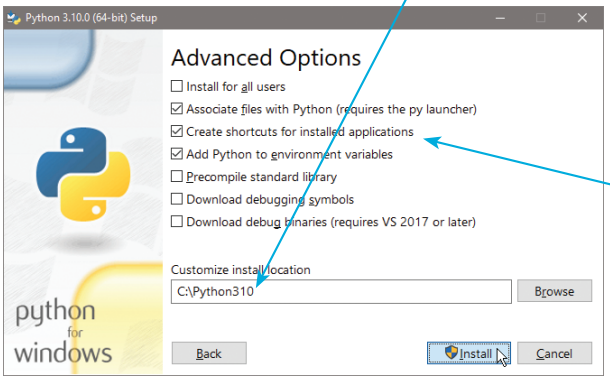
3. Next, click the "Customize installation" option

4. Check all "Optional Features" options, then click **Next**

Be sure to check the option to install "pip" so you can easily install Python packages later.

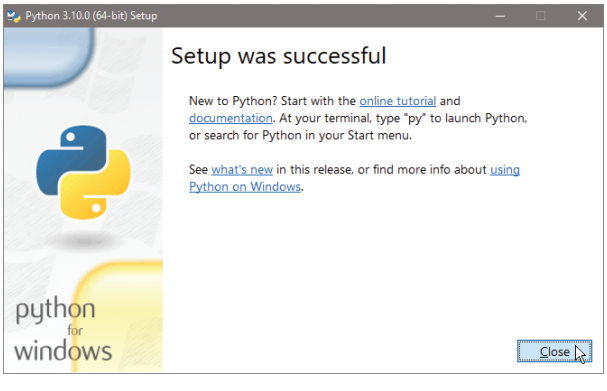


5 Now, change the lengthy suggested installation location to a more simple location of "C:\Python310"

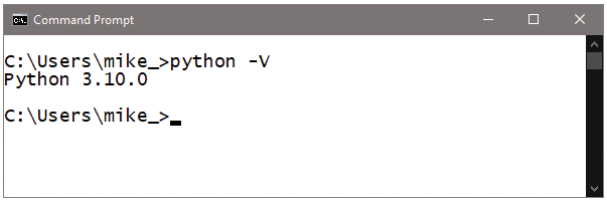


Accept the suggested features in the Advanced Options dialog – as illustrated here.

6 Click on **Install** to begin copying files onto your PC, then click the **Close** button to complete the installation



7 To confirm Python is now available, launch a Command Prompt window (run **cmd.exe**) and precisely enter the command **python -V** – the Python interpreter should respond with its version number



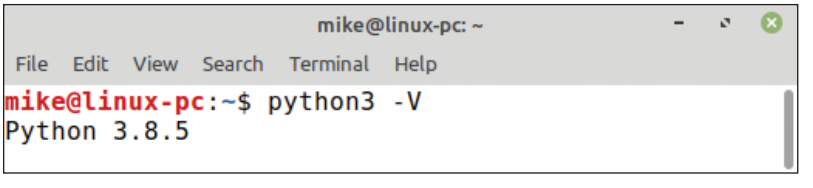
The letter V in the command must be uppercase. Ensure the command responds with the version number before proceeding to the examples in this book.

Installing Python on Linux

Linux distributions will, typically, include a version of Python. If installed, the Python interpreter can be invoked from a Linux Terminal prompt with the command **python** for the old 2.x series, and with the command **python3** for the supported 3.x series.

Modern Linux distros ship with the Python 3.x series, but not necessarily with the latest version released. You can, however, find more recent versions in the "deadsnakes" PPA (Personal Package Archive), and the latest version can be easily installed using the **sudo** command and the **apt** (Advanced Package Tool) utility:

1. Launch a Terminal window and precisely enter this command to reveal the currently installed Python version
   python3 -V



2. Enter these commands to install PPA management tools and authenticity key tools
   sudo apt install software-properties-common
   sudo apt install ca-certificates

3. Next, enter this command to add the "deadsnakes" PPA to your sources list
   sudo add-apt-repository ppa:deadsnakes/ppa

4. Hit **Enter** when asked if you wish to continue

5. Now, enter this command to update your sources list
   sudo apt update

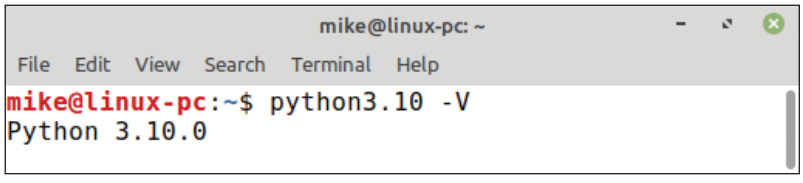
6. Then, enter this command to install the latest Python release – at the time of writing, that's version 3.10
   sudo apt install python3.10

7. Hit **Y** (yes) when asked if you wish to continue

These instructions are demonstrated using the popular Linux Mint operating system. You can discover more about Linux Mint from the companion book in this series entitled Linux in easy steps (visit www.ineasysteps.com for more information).

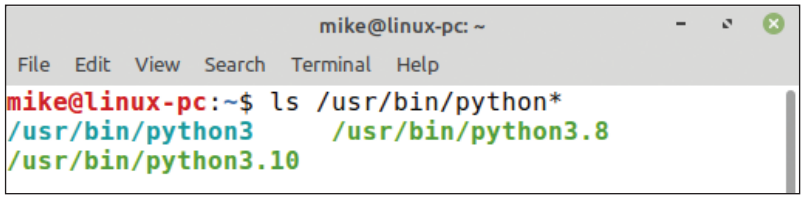
If you receive a "No protocol specified" error, enter the command **xhost + local:** and try again.

8 Confirm that the installation was successful by entering this command
python3.10 -V



The **python3** command will continue to invoke the default version that was included in your Linux distro.

9 Now, enter a command to list all the installed versions of Python on your system
ls /usr/bin/python*

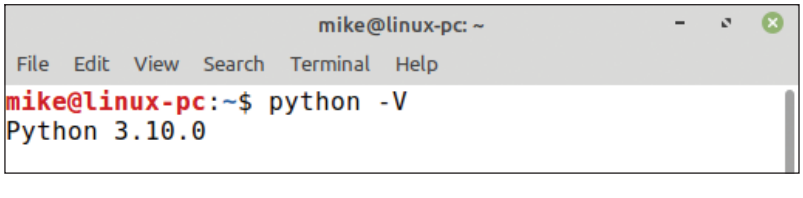


See that the **python3** command is simply a link to invoke the 3.8 version here and, as there is no 2.x series installed, you can create an alias for the **python** command to invoke the later 3.10 version:

10 Open your **.bashrc** configuration file in a text editor – such as in the **nano** text editor with this command
nano ~/.bashrc

11 Add this line at the end of the **.bashrc** configuration file, then save the file and close the editor
alias python=/usr/bin/python3.10

12 Open a new Terminal window and enter a **python -V** command to confirm **python** is invoking the latest version

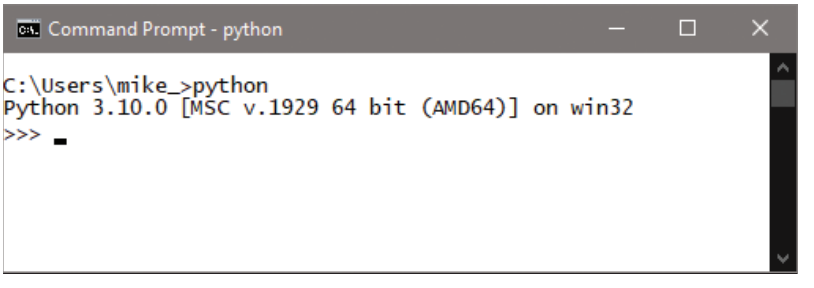


The Python programming examples in this book are all created in a plain text editor such as **nano**.

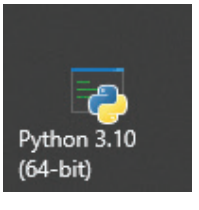
Meeting the interpreter

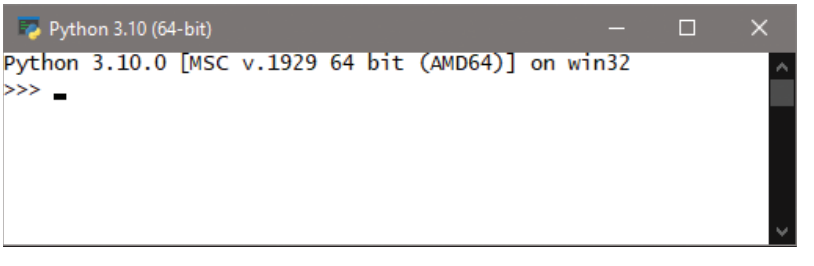
The Python interpreter processes text-based program code, and also has an interactive mode where you can test snippets of code and is useful for debugging code. Python's interactive mode can be entered in a number of ways:

- From a regular Command Prompt – simply enter the command **python** to produce the Python primary prompt **>>>** where you can interact with the interpreter.

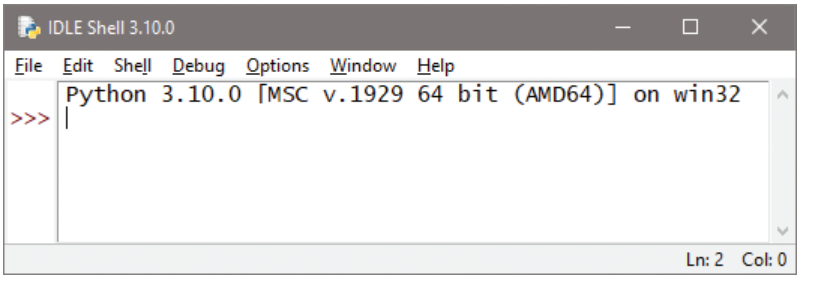


- From the Start Menu – choose "Python" to open a window containing the Python **>>>** primary prompt.





- From the Start Menu – choose "IDLE" to launch a Python Shell window containing the Python **>>>** primary prompt.



Irrespective of the method used to enter interactive mode, the Python interpreter will respond in the same way to commands entered at its >>> primary prompt. In its simplest form, the interpreter can be used as a calculator.

1. Enter Python interactive mode, using any method outlined opposite, then type a simple addition and hit Return to see the interpreter print out the sum total

The Python interpreter also understands expressions, so parentheses can be used to give higher precedence – the part of the expression enclosed within parentheses will be calculated first.

2. Next, at the Python prompt enter an expression with three components without specifying any precedence order

3. Now, at the Python prompt enter the same expression but add parentheses to specify precedence order

Spaces in expressions are ignored, so 8+4 can also be entered with added spaces for clarity – as illustrated here.

Interactive mode is mostly used to test snippets of code and for debugging code.

"IDLE" is an acronym for Python's Integrated DeveLopment Environment, but has limited features so is not used to demonstrate examples in this book.

Writing your first program

Python's interactive mode is useful as a simple calculator, but you can create programs for more extensive functionality. A Python program is simply a plain text file script created with an editor, such as Windows' Notepad, that has been saved with a ".py" file extension. Python programs can be executed by stating the script file name after the **python** command at a terminal prompt.

The traditional first program to create when learning any programming language simply prints out a specified greeting message. In Python, the **print()** function is used to specify the message within its parentheses. This must be a string of characters enclosed between quote marks. These may be " " double quote marks or ' ' single quote marks – but not a mixture of both.

Don't use a word processor to create program files as they add format information to the file.

hello.py

1. On Windows, launch any plain text editor such as the Notepad application

2. Next, precisely type the following statement into the empty text editor window
   print('Hello World!')

3. Now, create a new directory at **C:\MyScripts** and save the file in it as **hello.py**

4. Finally, launch a Command Prompt window, navigate to the new directory and precisely enter the command **python hello.py** – to see the Python interpreter run your program and print out the specified greeting message

Windows' Notepad can be launched as follows: press **Winkey** + **R** on the keyboard to open a Run dialog, type "notepad" into the Run dialog, then hit **Enter** to launch the Notepad app.

The directory created at **C:\MyScripts** will be used to contain all Windows examples in this book.

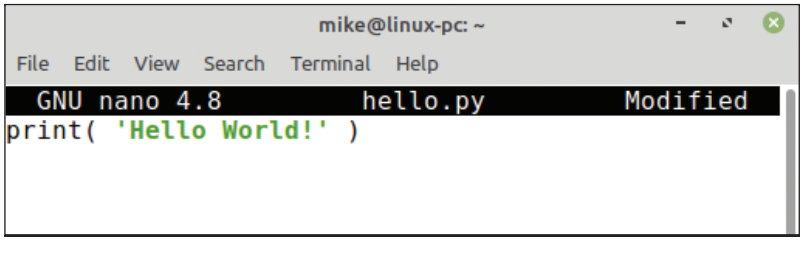
The procedure to create the traditional first Python program is identical on Linux systems to that on Windows systems. It is, however, important to be aware, on any platform where different versions of Python are installed: you must use the correct command to call upon the particular Python interpreter required. This is especially important on Linux systems that still have the Python 2.x series installed as their default. This means that the command **python** will assume you want to call the old interpreter. Where Python 3.x series is installed, and you want to call that particular interpreter to process a script, you must ensure that the command **python** will call upon the later version's interpreter.

1. On Linux, launch any plain text editor such as the **nano** application

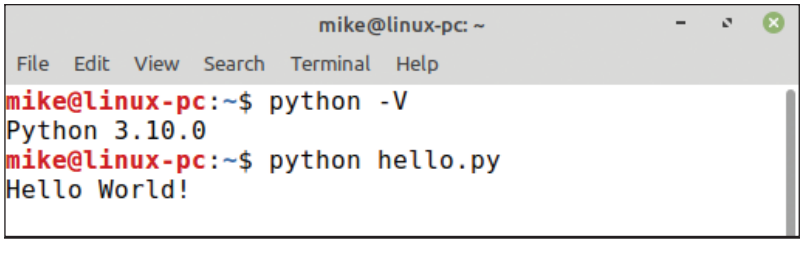
2. Next, precisely type the following statement into the empty text editor window
   `print( 'Hello World!' )`

3. Now, save the file in your home directory as **hello.py**

hello.py



4. Finally, launch a Terminal window and navigate to your home directory and enter the command **python hello.py** – to see the Python interpreter run your program and print out the specified greeting message



All further examples in this book are illustrated on Windows (simply because that platform has the most users) but they can also be created and executed on Linux.

Employing variables

In programming, a "variable" is a container in which a data value can be stored within the computer's memory. The stored value can then be referenced using the variable's name. The programmer can choose any name for a variable, except the Python keywords listed on the inside front cover of this book, and it is good practice to choose meaningful names that reflect the variable's content.

Data to be stored in a variable is assigned in a Python program declaration statement with the = assignment operator. For example, to store the numeric value eight in a variable named "a":

```
a = 8
```

The stored value can then be referenced using the variable's name, so that the statement **print(a)** will output the stored value **8**. That variable can subsequently be assigned a different value, so its value can vary as the program proceeds – hence the term "variable".

In Python programming, a variable must be assigned an initial value ("initialized") in the statement that declares it in a program – otherwise the interpreter will report a "not defined" error.

Multiple variables can be initialized with a common value in a single statement using a sequence of = assignments. For example, to initialize variables named "a", "b" and "c", each with a numeric value of eight, like this:

```
a = b = c = 8
```

Alternatively, multiple variables can be initialized with differing values in a single statement using comma separators. For example, to initialize variables named "a", "b" and "c" with numeric values of one, two and three respectively, like this:

```
a , b , c = 1 , 2 , 3
```

Some programming languages, such as Java, demand you specify what type of data a variable may contain in its declaration. This reserves a specific amount of memory space and is known as "static typing". Python variables, on the other hand, have no such limitation and adjust the memory allocation to suit the various data values assigned to their variables ("dynamic typing"). This means they can store integer whole numbers, floating-point numbers, text strings, or boolean values of **True** or **False** as required.

String data must be enclosed within quote marks to denote the start and end of the string.

Programming languages that require variable types to be specified are alternatively known as "strongly typed" whereas those that do not are alternatively known as "loosely typed".

Optionally, comments can be added to your Python scripts to describe the purpose of statements or sections of code if preceded by a # hash character. Everything following the # hash character up to the end of the line is ignored by the Python interpreter. It is useful to comment your code to make its purpose clear to others or when revisiting the code yourself later.

1. Launch a plain text editor, then declare and initialize a variable – then display its stored value

```
# Initialize a variable with an integer value.
var = 8
print( var )
```

var.py

2. Next, assign a new value and display that stored value

```
# Assign a float value to the variable.
var = 3.142
print( var )
```

3. Now, assign a different value and display the stored value

```
# Assign a string value to the variable.
var = 'Python in easy steps'
print( var )
```

4. Finally, assign another value and display the stored value

```
# Assign a boolean value to the variable.
var = True
print( var )
```

5. Save the file in your scripts directory, then open a Command Prompt window there and run the program – to see the stored values output as the program proceeds

```
Command Prompt

C:\MyScripts>python var.py
8
3.142
Python in easy steps
True

C:\MyScripts>
```

Multi-line comments can be added to a script if enclosed between triple quote marks """…""" .

Obtaining user input

Just as a data value can be assigned to a variable in a Python script, a user-specified value can be assigned to a variable with the Python **input()** function. This accepts a string within its parentheses that will prompt the user for input by displaying that string then wait to read a line of input.

User input is read as a text string, even when it's numeric, and can be assigned to a variable using the **=** assignment operator as usual. The value assigned to any variable can be displayed by specifying the variable name to the **print()** function – to reference that variable's stored value.

Multiple values to be displayed can be specified to the **print()** function as a comma-separated list within its parentheses.

input.py

1. Launch a plain text editor, then declare and initialize a variable by requesting user input

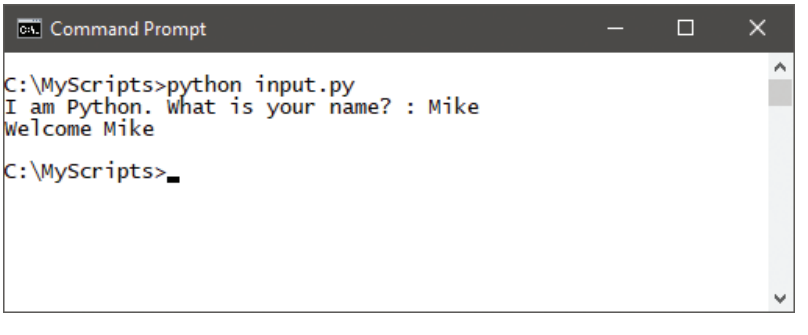
```
# Initialize a variable with a user-specified value.
user = input( 'I am Python. What is your name? : ' )
```

2. Next, display a response message confirming the input by referencing the stored user name

```
# Output a string and a variable value.
print( 'Welcome' , user )
```

3. Now, save the file in your scripts directory then open a Command Prompt window there and run this program – enter your name, then hit Return to see the response message include your name

Notice that the prompt string ends with a space that is displayed in output – so the user entry is separated from the colon when typed in.



When multiple values are specified to the **print()** function it will display each value in output separated by a single space by default. An alternative separator can, however, be specified by adding a **sep** parameter to the comma-separated list. For example, **sep = '*'** will display each value in output separated by an asterisk character.

Output displayed by the **print()** function will, by default, add an invisible **\n** newline character at the end of the line to automatically move the print head to the next line. An alternative line ending can, however, be specified by adding an **end** parameter to the comma-separated list. For example, **end = '!'** will display each value in output then end the line with an exclamation mark.

You can explicitly specify a newline to the **end** parameter; for example, **end='!\n'** adds both an exclamation mark and a newline character.

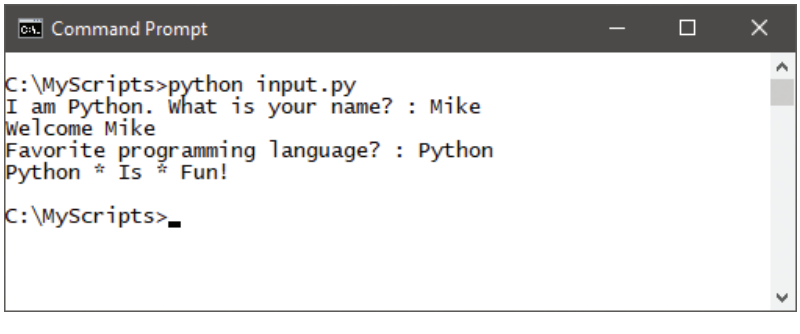
4 Edit the script to declare and initialize a second variable by requesting more user input

```
# Initialize another variable with a user-specified value.
lang = input( 'Favorite programming language? : ' )
```

5 Next, display a response message confirming the input by referencing the stored language name – and specifying a custom separator and a custom line ending

```
# Output a string and a variable value.
print( lang , 'Is' , 'Fun' , sep = ' * ' , end = '!\n' )
```

6 Now, save the file once more, then open a Command Prompt window there and run this program again – enter your name and a programming language, then hit Return to see the response message include your user input

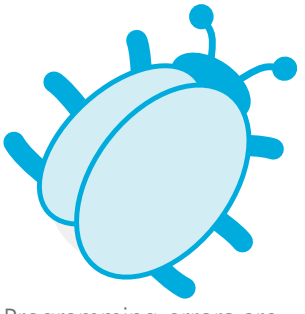


You can include space characters around the separator character for clarity – like those shown around the asterisk character in this example.

Correcting errors

In Python programming there are three types of error that can occur. It is useful to recognize the different error types so they can be corrected more easily:

- **Syntax Error** – occurs when the interpreter encounters code that does not conform to the Python language rules. For example, a missing quote mark around a string. The interpreter halts and reports the error without executing the program.
- **Runtime Error** – occurs during execution of the program, at the time when the program runs. For example, when a variable name is later mis-typed so the variable cannot be recognized. The interpreter runs the program but halts at the error and reports the nature of the error as an "Exception".
- **Semantic Error** – occurs when the program performs unexpectedly. For example, when order precedence has not been specified in an expression. The interpreter runs the program and does not report an error.



Programming errors are often called "bugs" and the process of tracking them down is often called "debugging".

Correcting syntax and runtime errors is fairly straightforward, as the interpreter reports where the error occurred or the nature of the error type, but semantic errors require code examination.

syntax.py

1. Launch a plain text editor then add a statement to output a string that omits a closing quote mark

```
print( 'Python in easy steps )
```

2. Save the file in your scripts directory then open a Command Prompt window there and run this program – to see the interpreter report the syntax error and indicate the position in the code where the error occurs

Typically, the syntax error indicator points to the next character after an omission in the code.

```
Command Prompt

C:\MyScripts>python syntax.py
  File "syntax.py", line 1
    print( 'Python in easy steps  )
                                  ^
SyntaxError: unterminated string literal (detected at line 1)

C:\MyScripts>_
```

3 Insert a quote mark before the closing parenthesis to terminate the string, then save the file and run the program again – to see the error has been corrected

4 Next, begin a new program by initializing a variable, then try to output its value with an incorrect variable name – to see the interpreter report a runtime error

```
title = 'Python in easy steps'
print( titel )
```

runtime.py

```
Command Prompt

C:\MyScripts>python runtime.py
Traceback (most recent call last):
  File "runtime.py", line 2, in <module>
    print( titel )
NameError: name 'titel' is not defined. Did you mean: 'title'

C:\MyScripts>
```

Hot tip

Details of how to handle runtime Exception errors in your script code are provided on page 74.

5 Amend the variable name to match that in the variable declaration, then save the file and run the program again – to see the error has been corrected

6 Now, begin a new program by initializing a variable, then try to output an expression using its value without explicit precedence – to see a possibly unexpected result of 28

```
num = 3
print( num * 8 + 4 )
```

semantic.py

```
Command Prompt

C:\MyScripts>python semantic.py
28

C:\MyScripts>
```

7 Add parentheses to group the expression as `3 * ( 8 + 4 )`, then save the file and run the program again – to see the expected result of 36, correcting the semantic error

Summary

- Python is a high-level programming language that is processed by the Python interpreter to produce results.
- Python uses indentation to group statements into code blocks, where other languages use keywords or punctuation.
- Python 2.7 was the final version of the 2.x series of development, but the 3.x series has the latest improvements.
- Windows users can install Python with an installer, and Linux users can install Python with their package manager.
- The Python interpreter has an interactive mode where you can test snippets of code and is useful for debugging code.
- A Python program is simply a text file created with a plain text editor and saved with a ".py" file extension.
- The Python **print()** function outputs the string specified within its parentheses.
- String values must be enclosed between quote marks.
- Where multiple versions of Python are installed on the same system it is important to explicitly call the desired interpreter.
- A Python variable is a named container whose stored value can be referenced via that variable's name.
- A Python variable can contain any data type but must be given an initial value when it is declared.
- The Python **input()** function outputs the string specified within its parentheses, then waits to read a line of input.
- Syntax errors due to incorrect code are recognized by the interpreter before execution of the program.
- Runtime errors due to exceptions are recognized by the interpreter during execution of the program.
- Semantic errors due to unexpected performance are not recognized by the interpreter.