Setting up

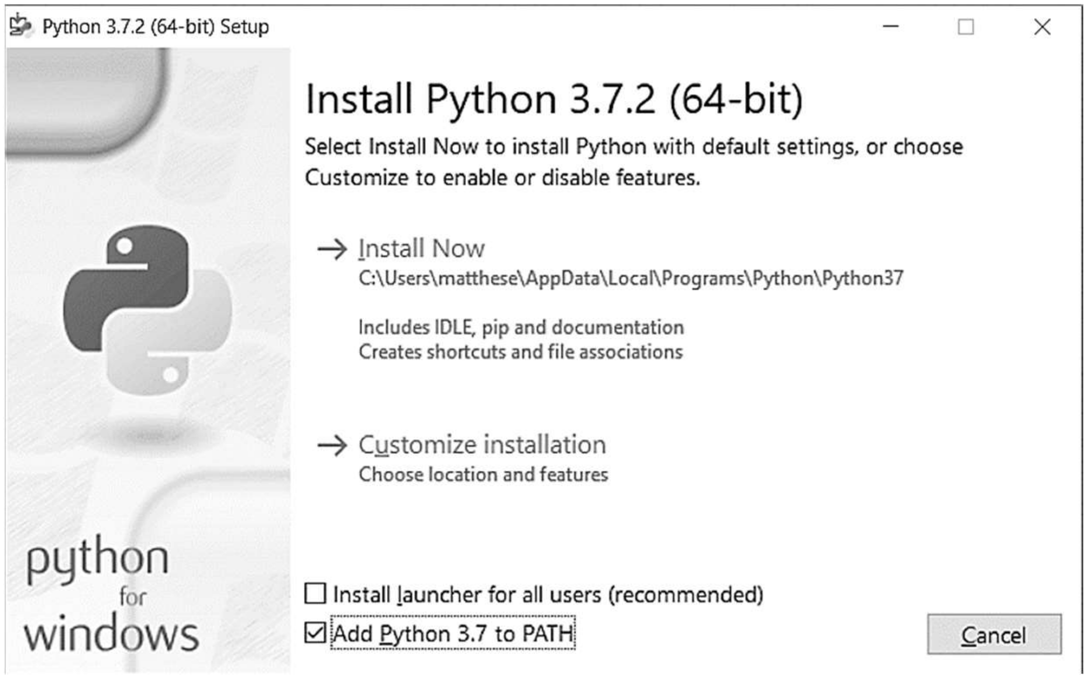
K.N. Toosi

Fall 2023

Setting Up Your Programming Environment

- Python is a **cross-platform** programming language, which means it runs on all the major operating systems.
- Installing Python for Windows:
  1. Go to https://python.org/ and hover over the Downloads link. Download the latest version of Python.
  2. Run the installer. **Make sure you select the option Add Python to PATH**, which will make it easier to configure your system correctly.
  3. Open a command window and enter python in lowercase. See a Python prompt (>>>), which means Windows has found the version of Python you just installed.

- Any time you want to run a snippet of Python code, open a command window and start a Python terminal session. To close the terminal session enter the command exit().


Python 3.7.2 (64-bit) Setup

Install Python 3.7.2 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\matthese\AppData\Local\Programs\Python\Python37

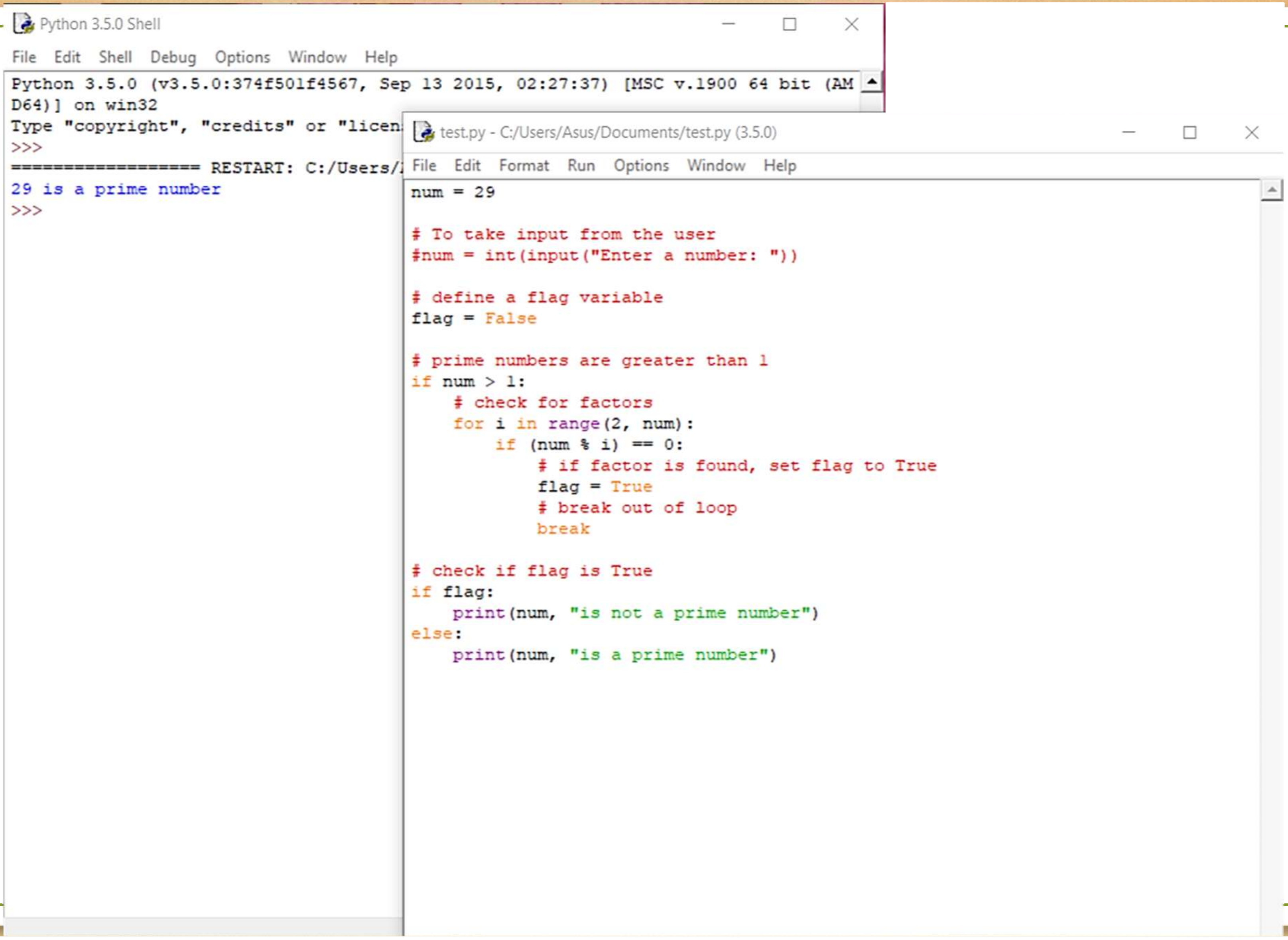
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☐ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

Python IDLE

- **IDLE** is "Integrated Development and Learning Environment".
- The Python installer for Windows contains the IDLE module by default. IDLE can be used to execute a single statement and also to create, modify, and execute Python scripts.
- The python IDLE has two main windows: Shell window and Editor.
  - Python Shell is interactive, similar to command prompt but with additional features of syntax highlighting and automatic indentation.
  - Editor Window can contain multiple lines of code and can be saved as a file.
- Default IDLE is sufficient for beginners.
- But for higher levels, more professional tools are needed (**IDE**). An IDE normally consolidates source code editor, build automation tools and a debugger. E.g. of well-known Python IDEs:
  - PyCharm
  - Spyder
  - VSCode



Python 3.5.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "licen
>>>
================== RESTART: C:/Users/
29 is a prime number
>>>
```

test.py - C:/Users/Asus/Documents/test.py (3.5.0)
File Edit Format Run Options Window Help

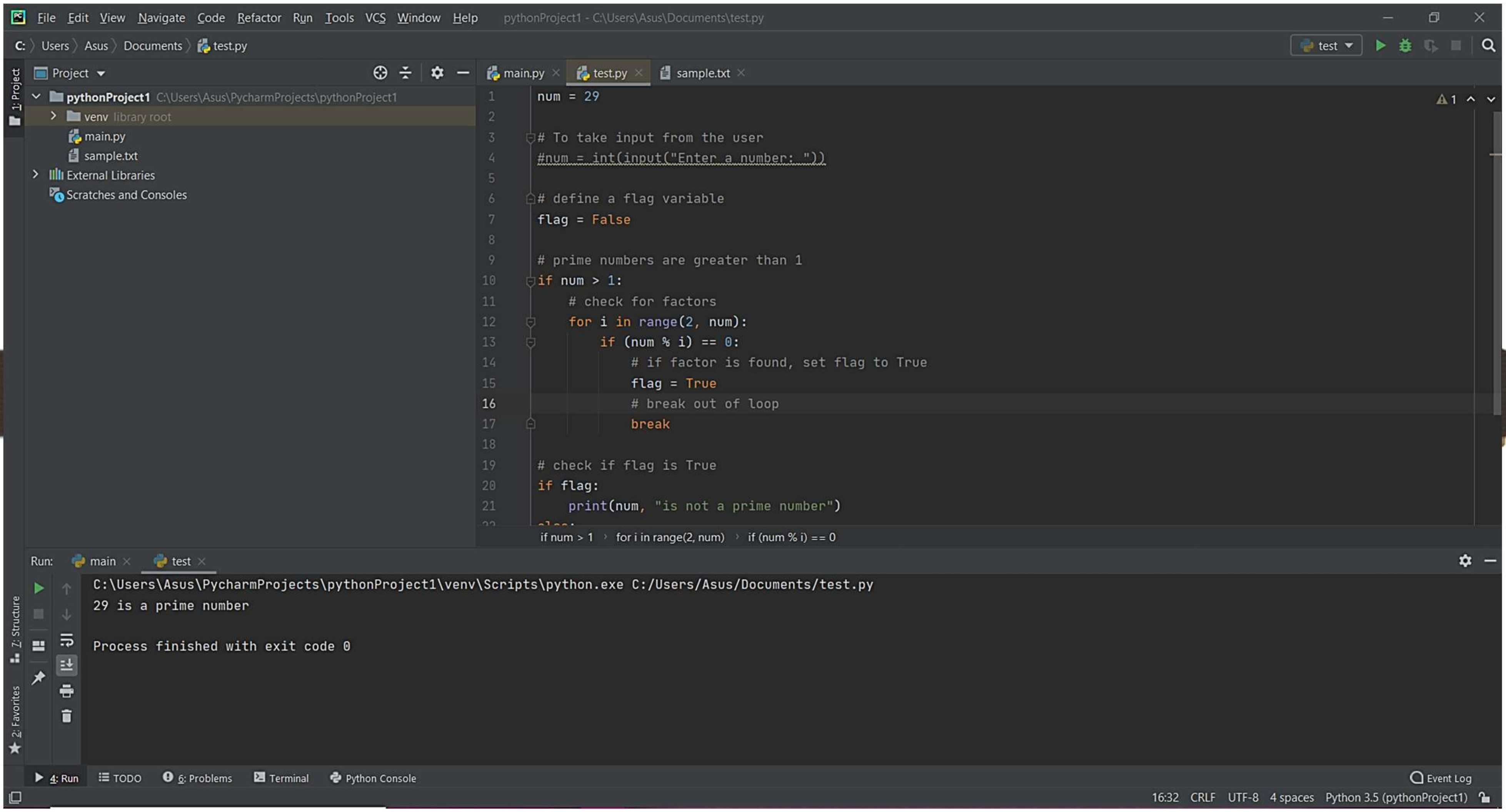
```python
num = 29

# To take input from the user
#num = int(input("Enter a number: "))

# define a flag variable
flag = False

# prime numbers are greater than 1
if num > 1:
    # check for factors
    for i in range(2, num):
        if (num % i) == 0:
            # if factor is found, set flag to True
            flag = True
            # break out of loop
            break

# check if flag is True
if flag:
    print(num, "is not a prime number")
else:
    print(num, "is a prime number")
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
pythonProject1 - C:\Users\Asus\Documents\test.py
C: Users Asus Documents test.py
Project
pythonProject1 C:\Users\Asus\PycharmProjects\pythonProject1
venv library root
main.py
sample.txt
External Libraries
Scratches and Consoles
main.py
test.py
sample.txt
if num > 1 › for i in range(2, num) › if (num % i) == 0
Run: main test
4: Run TODO 6: Problems Terminal Python Console Event Log
16:32 CRLF UTF-8 4 spaces Python 3.5 (pythonProject1)

```python
num = 29

# To take input from the user
#num = int(input("Enter a number: "))

# define a flag variable
flag = False

# prime numbers are greater than 1
if num > 1:
    # check for factors
    for i in range(2, num):
        if (num % i) == 0:
            # if factor is found, set flag to True
            flag = True
            # break out of loop
            break

# check if flag is True
if flag:
    print(num, "is not a prime number")
```

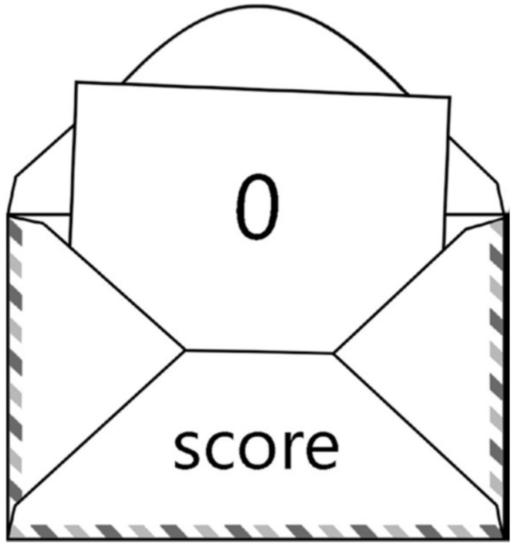
```
C:\Users\Asus\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:/Users/Asus/Documents/test.py
29 is a prime number

Process finished with exit code 0
```

Variables

Variables

- In programming, we need to remember and manipulate data all the time. In order to *store* and *manipulate* data, we use a variable.
- Definition: A **variable** is a named for memory location that holds a value.
- The contents of a variable can change or vary over time; this is why it's called a variable.
- Another way to think of a variable is as an envelope or a box into which you can put a value. In this way of thinking, a variable is a container—a storage space—with a name. The contents are the value. The name never changes, but the contents can change over time.



Assignment Statement

- In Python, you create and give a value to a variable with an assignment statement.
- Definition: An **assignment statement** is a line of code in which a variable is given a value.
- Example:
  - age = 29
  - name = 'Fred'
  - alive = True
  - gpa = 3.9

Types of data

- The four basic types of data are called **intege**r numbers, **floating-point** numbers, **strings**, and **Booleans**.
  - Integers: Integer numbers (or simply, integers) are counting numbers, like 1, 2, 3, but also include 0 and negative numbers. E.g., number of people in a room.
    - 23, 0, -1, 10
  - Floats: Floating-point numbers (or simply floats) are numbers that have a decimal point in them. E.g., grade point average.
    - 1.5, 0.0, 4.57, -3.21
  - Strings: Strings (also called text) are any sequences of characters. E.g., name, address, course name.
    - "Joe", 'This is some string data'
  - Booleans: Booleans are a type of data that can only have one of two values: True or False. E.g., the state of a light switch: True for on, False for off
    - True, False
- Can we convert data types? Yes, more on this later!

Type( )

- A value is one of the basic things a program works with, like a letter or a number. Some values we have seen so far are 2, 42.0, and 'Hello, World!'
- These values belong to different types: 2 is an integer, 42.0 is a floating-point number, and 'Hello, World!' is a string, so-called because the letters it contains are strung together.
- If you are not sure what type a value has, the interpreter can tell you:

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Hello, World!')
<class 'str'>
```

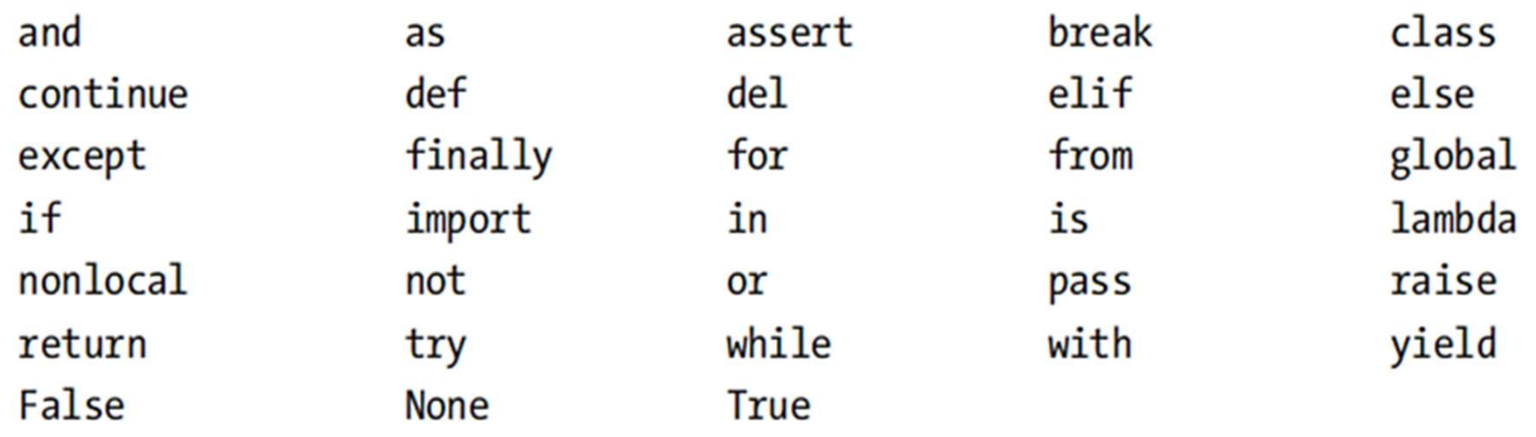
- In these results, the word "class" is used in the sense of a category; a type is a category of values.

Rules for naming variables

- When you're using variables in Python, you need to adhere to a few rules and guidelines. Breaking some of these rules will cause errors:
  - Variable names can contain only *letters, numbers, and underscores*. They can start with a letter or an underscore, but not with a number.
  - Spaces are not allowed in variable names, but underscores can be used to separate words in variable names.
  - Avoid using Python keywords and function names as variable names; that is, do not use words that Python has reserved for a particular programmatic purpose, such as the word print.
  - Variable names should be short but descriptive. For example, name is better than n, student_name is better than s_n, and name_length is better than length_of_persons_name.
  - Be careful when using the lowercase letter l and the uppercase letter O because they could be confused with the numbers 1 and 0.
  - Cannot contain math symbols (+, -, /, *, %, parentheses)

Don’t use keywords as variable name

- You cannot use any of these words as a variable name. If you attempt to do so, the Python compiler might generate an error message when it tries to compile your program.



and	as	assert	break	class
continue	def	del	elif	else
except	finally	for	from	global
if	import	in	is	lambda
nonlocal	not	or	pass	raise
return	try	while	with	yield
False	None	True		

Case sensitive

- Variable names and keywords in Python are all case sensitive. For example:
    - a variable named myVariable is not the same as one named one named myvariable and is not the same as one named MyVariable
    - print is not the same as Print

Mathematicians be aware!

- The equals sign in an assignment statement is very different from the way an equals sign is used in math.
- consider these two lines of code:

```
myAge = 25
myAge = myAge + 1
```

- This is not an equation; it is an assignment statement. Here is what the second line says:
    1. Take the current value of the variable myAge
    2. Add 1 to it
    3. Put the resulting value back in to the variable myAge
- This statement effectively changes the value of the variable myAge by adding 1 to it.

Tips and tricks

- **Multiple Assignment**
  - You can assign values to more than one variable using just a single line. For example, here's how you can initialize the variables x, y, and z to zero:

```
>>> x, y, z = 0, 0, 0
```

- **Increment & decrement**
  - x = x + 2 → x += 2
  - x = x - 2 → x -= 2
- **White Space**
  - **For now**, all of our statements must start in column 1 of each line, but you can add as many space characters as you want in between items on a line.

Constants

- A constant is like a variable whose value stays the same throughout the life of a program. Python doesn’t have built-in constant types, but Python programmers use all capital letters to indicate a variable should be treated as a constant and never be changed:
    - MAX_CONNECTIONS = 5000

Simple math – Order of operations

- Python recognizes the following set of math operators:

  + Add

  - Subtract

  * Multiply

  / Divide

  // Integer Divide

  ** Raise to the power of

  % Modulo (also known as remainder)

  ( ) Grouping

- some math operators had precedence over other ones. For example, look at this assignment statement:

  x = 5 + 4 * 9 / 6

- the precedence order as follows:

  1. Parentheses
  2. Exponents
  3. Multiplication
  4. Division
  5. Addition
  6. Subtraction

- Tip: Adding parentheses as in the preceding statements makes your intent much clearer

Comment

When you are writing software, you wind up making a lot of decisions about how you approach different problems.You may want to explain to the reader (who could be a future version of you, or someone else) why you did something the way you wound up doing it, or how some intricate piece of code works.

Documentation like that, written directly in your code, is called a comment. **Comments are completely ignored by Python; they are only there for humans.**

Three ways to write comments in Python:

- full-line comment,
- add a comment after a line of code,
- use a multiline comment.

- **Full-Line Comment:** Start a line with the # character, followed by your comment:

```
# This whole line is a comment
```

- **Add a Comment After a Line of Code:** You can put a comment at the end of a line of code to explain what is going on inside that line.

```
score = 0 # Initializing the score
priceWithTax = price * 1.09    # add in 9% tax
```

- **Multiline comment:** You can create a comment that spans any number of lines by making one line with three quote marks (single or double quotes), writing any number of comment lines, and ending with the same three quote characters (single or double quotes) on the last line.

```
'''
A multiline comment starts with a line of three quote
characters (above) This is a long comment block
It can be any length
'''
```

Errors:

1. **Syntax error**: “Syntax” refers to the structure of a program and the rules about that structure. For example, parentheses have to come in matching pairs, so (1 + 2) is legal, but 8) is a syntax error. **If there is a syntax error anywhere in your program**, Python displays an error message and quits, and **you will not be able to run the program**.
2. **Runtime error:** The second type of error is a runtime error, so called because **the error does not appear until after the program has started running**. These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.
3. **Semantic error:** The third type of error is “semantic”, which means related to meaning. If there is a semantic error in your program**, it will run without generating error messages**, but **it will not do the right thing**. It will do something else. Specifically, it will do what you told it to do. Identifying semantic errors can be tricky because it requires you to work backward by looking at the output of the program and trying to figure out what it is doing.

Example of Run-time error

- Overflow

  -
    ```
    a = 10**1000
    b = a + 0.1
    print(b)
    ```
- Zero division

  -
    ```
    a = 10
    b = 0
    print (a/b)
    ```