# HELLO WORLD

## Welcome

Python is a programming language. Like other languages, it gives us a way to communicate ideas. In the case of a programming language, these ideas are “commands” that people use to communicate with a computer!

We convey our commands to the computer by writing them in a text file using a programming language. These files are called *programs*. Running a program means telling a computer to read the text file, translate it to the set of operations that it understands, and perform those actions.

## Comments

Ironically, the first thing we’re going to do is show how to tell a computer to ignore a part of a program. Text written in a program but not run by the computer is called a [comment](https://www.codecademy.com/resources/docs/python/comments?page_ref=catalog). Python interprets anything after a # as a comment.

Comments can:

* Provide context for why something is written the way it is:

# This variable will be used to count the number of times anyone tweets the word persnickety  
persnickety\_count = 0

* Help other people reading the code understand it faster:

# This code will calculate the likelihood that it will rain tomorrow  
complicated\_rain\_calculation\_for\_tomorrow()

* Ignore a line of code and see how a program will run without it:

# useful\_value = old\_sloppy\_code()  
useful\_value = new\_clean\_code()

## Print

Now what we’re going to do is teach our computer to communicate. The gift of speech is valuable: a computer can answer many questions we have about “how” or “why” or “what” it is doing. In Python, the [print()](https://www.codecademy.com/resources/docs/python/built-in-functions/print?page_ref=catalog) function is used to tell a computer to talk. The message to be printed should be surrounded by quotes:

# from Mary Shelley's Frankenstein  
print("There is something at work in my soul, which I do not understand.")

In the above example, we direct our program to print() an excerpt from a notable book. The printed words that appear as a result of the print() function are referred to as *output*. The output of this example program would be:

There is something at work in my soul, which I do not understand.

## Strings

Computer programmers refer to blocks of text as [strings](https://www.codecademy.com/resources/docs/python/strings?page_ref=catalog). In our last exercise, we created the string “Hello world!”. In Python a string is either surrounded by double quotes ("Hello world") or single quotes ('Hello world'). It doesn’t matter which kind you use, just be consistent.

## Variables

Programming languages offer a method of storing data for reuse. If there is a greeting we want to present, a date we need to reuse, or a user ID we need to remember we can create a [variable](https://www.codecademy.com/resources/docs/python/variables?page_ref=catalog) which can store a value. In Python, we [*assign*](https://www.codecademy.com/resources/docs/python/operators?page_ref=catalog) variables by using the equals sign (=).

message\_string = "Hello there"  
# Prints "Hello there"  
print(message\_string)

In the above example, we store the message “Hello there” in a variable called message\_string. Variables can’t have spaces or symbols in their names other than an underscore (\_). They can’t begin with numbers but they can have numbers after the first letter (e.g., cool\_variable\_5 is OK).

It’s no coincidence we call these creatures “variables”. If the context of a program changes, we can update a variable but perform the same logical process on it.

# Greeting  
message\_string = "Hello there"  
print(message\_string)  
   
# Farewell  
message\_string = "Hasta la vista"  
print(message\_string)

Above, we create the variable message\_string, assign a welcome message, and print the greeting. After we greet the user, we want to wish them goodbye. We then update message\_string to a departure message and print that out.

## Errors

Humans are prone to making mistakes. Humans are also typically in charge of creating computer programs. To compensate, programming languages attempt to understand and explain mistakes made in their programs.

Python refers to these mistakes as [errors](https://www.codecademy.com/resources/docs/python/errors?page_ref=catalog) and will point to the location where an error occurred with a ^ character. When programs throw errors that we didn’t expect to encounter we call those errors *bugs*. Programmers call the process of updating the program so that it no longer produces unexpected errors *debugging*.

Two common errors that we encounter while writing Python are SyntaxError and NameError.

* SyntaxError means there is something wrong with the way your program is written — punctuation that does not belong, a command where it is not expected, or a missing parenthesis can all trigger a SyntaxError.
* A NameError occurs when the Python interpreter sees a word it does not recognize. Code that contains something that looks like a variable but was never defined will throw a NameError.

## Numbers

Computers can understand much more than just strings of text. Python has a few [numeric *data types*.](https://www.codecademy.com/resources/docs/python/data-types?page_ref=catalog) It has multiple ways of storing numbers. Which one you use depends on your intended purpose for the number you are saving.

An *integer*, or int, is a whole number. It has no decimal point and contains all counting numbers (1, 2, 3, …) as well as their negative counterparts and the number 0. If you were counting the number of people in a room, the number of jellybeans in a jar, or the number of keys on a keyboard you would likely use an integer.

A *floating-point number*, or a float, is a decimal number. It can be used to represent fractional quantities as well as precise measurements. If you were measuring the length of your bedroom wall, calculating the average test score of a seventh-grade class, or storing a baseball player’s batting average for the 1998 season you would likely use a float.

Numbers can be assigned to variables or used literally in a program:

an\_int = 2  
a\_float = 2.1  
   
print(an\_int + 3)  
# Output: 5

Above we defined an integer and a float as the variables an\_int and a\_float. We printed out the sum of the variable an\_int with the number 3. We call the number 3 here a *literal*, meaning it’s actually the number 3 and not a variable with the number 3 assigned to it.

Floating-point numbers can behave in some unexpected ways due to how computers store them. For more information on floating-point numbers and Python, review [Python’s documentation on floating-point limitations](https://docs.python.org/3/tutorial/floatingpoint.html).

## Calculations

Computers absolutely excel at performing calculations. The “compute” in their name comes from their historical association with providing answers to mathematical questions. Python performs the [arithmetic operations](https://www.codecademy.com/resources/docs/python/operators?page_ref=catalog) of addition, subtraction, multiplication, and division with +, -, \*, and /.

# Prints "500"  
print(573 - 74 + 1)  
   
# Prints "50"  
print(25 \* 2)  
   
# Prints "2.0"  
print(10 / 5)

Notice that when we perform division, the result has a decimal place. This is because Python converts all ints to floats before performing division. In older versions of Python (2.7 and earlier) this conversion did not happen, and integer division would always round down to the nearest integer.

Division can throw its own special error: ZeroDivisionError. Python will raise this error when attempting to divide by 0.

Mathematical operations in Python follow the standard mathematical [order of operations](https://www.codecademy.com/resources/docs/python/operators?page_ref=catalog).

## Data Types

Python is a strongly typed language, in the sense that at runtime it prevents typing errors and it engages in little implicit type conversion or [casting](https://www.codecademy.com/resources/docs/python/casting), i.e. converting one type to another without a specific call to a conversion function.

codecademy = 575

codecademy = "575 broadway"

After line 1, codecademy is an int. After line 2, codecademy is a str.

Python includes the following categories of built-in data types:

* String type: str
* Boolean type: bool
* Binary types: bytes, bytearray, memoryview
* Number types: int, float, complex
* Sequence Types: list, range, tuple
* Set types: set, frozenset
* Dictionary type: dict

### type()

The type() function can be used to retrieve the data type of an object:

message = "Hello, world!"

print(type(message))

# Output: <class 'str'>

### isinstance()

The isinstance() function can be used to test if an object is an instance of a specified type. This will print a boolean value for each function call, indicating if the object is an instance of the given type:

word = "purple"

languages = ("Python", "JavaScript", "Go")

print(isinstance(word, str)) # Output: True

print(isinstance(languages, list)) # Output: False

print(isinstance(languages, tuple)) # Ouput: True

## Casting

Casting, also known as type conversion, is a process that converts a variable’s data type into another data type. These conversions can be implicit (automatically interpreted) or explicit (using built-in functions).

### Implicit Type Conversion

The Python interpreter automatically performs type conversion on some operations without any user involvement.

Python avoids data loss by converting lower data types to higher data types. For example, an integer, 7, is converted to a float when added with another float, 2.2:

y = 7 + 2.2

# Python automatically type casts y into float

print(y)

# Output: 9.2

print(type(y))

# Output: <class 'float'>

Since the expression above represents the sum of two float values, the data type of y is also a float.

### Explicit Type Conversion

Explicit type casting involves Python’s predefined functions that act as a constructor of another data type:

* The str() function takes an integer or float as an argument and converts it to a string.
* The [int() function](https://www.codecademy.com/resources/docs/python/built-in-functions/int) takes a string or float as an argument converts it to an integer.
* The [float() function](https://www.codecademy.com/resources/docs/python/built-in-functions/float) takes an integer or string as an argument and converts it to a float.

### Operations on Different Types of Data

When operating on data, it is important to be mindful of the data types associated with it. The following code is a flawed attempt to print the square of a number specified by the user. When run, a TypeError will be thrown:

#### Code

#### Output

1

2

num = input("Please enter a number: ")

print(num \*\* 2)



Traceback (most recent call last):

File "main.py", line 1, in <module>

num = input("Please enter a number: ")

EOFError: EOF when reading a line

Run

The [input() function](https://www.codecademy.com/resources/docs/python/built-in-functions/input) takes input from the user and stores it in a variable as a string. However, the \*\* operator takes two numbers and returns the first number to the power of the second. To make the code work, the input variable must be cast to a number type. Try to edit the code above to successfully square a number inputted by the user.

## Operators

Operators are used to perform various operations on variables and values. The standard arithmetic and assignment operators are the most familiar.

### Syntax

The following code snippet uses the assignment operator, =, to set my\_variable to the value of num1 and num2 with an arithmetic operator acting on them. For example, if operator represented \*, my\_variable would be assigned a value of num1 \* num2.

my\_variable = num1 operator num2

Python operators can be organized into the following groups:

* Arithmetic operators for performing traditional math evaluations.
* Assignment operators for assigning values to variables.
* Comparison operators for comparing two values.
* Logical operators for combining boolean values.

### Arithmetic Operators

Python has the following arithmetic operators:

* Addition, +, which returns the sum of two numbers.
* Subtraction, -, which returns the difference of two numbers.
* Multiplication, \*, which returns the product of two numbers.
* Division, /, which returns the quotient of two numbers.
* Exponentiation, \*\*, which returns the value of one number raised to the power of another.
* Modulus, %, which returns the remainder of one number divided by another.
* Floor division, //, which returns the integer quotient of two numbers.

### Assignment Operators

Python includes the following assignment operators:

* The = operator assigns the value on the right to the variable on the left.
* The += operator updates a variable by incrementing its value and reassigning it.
* The -= operator updates a variable by decrementing its value and reassigning it.
* The \*= operator updates a variable by multiplying its value and reassigning it.
* The /= operator updates a variable by dividing its value and reassigning it.
* The %= operator updates a variable by calculating its modulus against another value and reassigning it.

### Comparison Operators

Python has the following comparison operators:

* Equal, ==, for returning True if two values are equal.
* Not equal, !=, for returning True if two values are not equal.
* Less than, <, for returning True if left value less than right value.
* Less than or equal to, <=, for returning True if left value is less than or equal to right value.
* Greater than, >, for returning True if left value greater than right value.
* Greater than or equal to, >=, for returning True if left value greater than or equal to right value.

### Logical Operators

Python has the following logical operators:

* The and operator returns True if both statements are True.
* The or operator returns True if either statement is True.
* The not operator returns True if its associated statement is False.

### Order of Operations

Python evaluates an expression in order of precedence as follows:

* Items in parentheses, ((…)), have the highest level of precedence, expressions within them are evaluated first.
* Exponentiation (\*\*)
* Multiplication and division operators (\*, /, // & %)
* Addition and subtraction (+ & -)
* Comparison (<, <=, > & >=)
* Equality (== & !=)
* not
* and
* or

**Note:** Items at the same precedence are evaluated left to right. The exception to this is exponentiation, which evaluates right to left.

## Changing Numbers

Variables that are assigned numeric values can be treated the same as the numbers themselves. Two variables can be added together, divided by 2, and multiplied by a third variable without Python distinguishing between the variables and *literals* (like the number 2 in this example). Performing arithmetic on variables does not change the variable — you can only update a variable using the = sign.

coffee\_price = 1.50  
number\_of\_coffees = 4  
   
# Prints "6.0"  
print(coffee\_price \* number\_of\_coffees)  
# Prints "1.5"  
print(coffee\_price)  
# Prints "4"  
print(number\_of\_coffees)  
   
# Updating the price   
coffee\_price = 2.00  
   
# Prints "8.0"  
print(coffee\_price \* number\_of\_coffees)  
# Prints "2.0"  
print(coffee\_price)  
# Prints "4"  
print(number\_of\_coffees)

We create two variables and assign numeric values to them. Then we perform a calculation on them. This doesn’t update the variables! When we update the coffee\_price variable and perform the calculations again, they use the updated values for the variable!

## Exponents

Python can also perform exponentiation. In written math, you might see an [exponent](https://www.codecademy.com/resources/docs/python/operators?page_ref=catalog) as a superscript number, but typing superscript numbers isn’t always easy on modern keyboards. Since this operation is so related to multiplication, we use the notation \*\*.

# 2 to the 10th power, or 1024  
print(2 \*\* 10)  
   
# 8 squared, or 64  
print(8 \*\* 2)  
   
# 9 \* 9 \* 9, 9 cubed, or 729  
print(9 \*\* 3)  
   
# We can even perform fractional exponents  
# 4 to the half power, or 2  
print(4 \*\* 0.5)

Here, we compute some simple exponents. We calculate 2 to the 10th power, 8 to the 2nd power, 9 to the 3rd power, and 4 to the 0.5th power.

## Modulo

Python offers a companion to the division operator called the modulo operator. The [modulo operator](https://www.codecademy.com/resources/docs/python/modulo?page_ref=catalog) is indicated by % and gives the remainder of a division calculation. If the number is divisible, then the result of the modulo operator will be 0.

# Prints 4 because 29 / 5 is 5 with a remainder of 4  
print(29 % 5)  
   
# Prints 2 because 32 / 3 is 10 with a remainder of 2  
print(32 % 3)  
   
# Modulo by 2 returns 0 for even numbers and 1 for odd numbers  
# Prints 0  
print(44 % 2)

Here, we use the modulo operator to find the remainder of division operations. We see that 29 % 5 equals 4, 32 % 3 equals 2, and 44 % 2 equals 0.

The modulo operator is useful in programming when we want to perform an action every nth-time the code is run. Can the result of a modulo operation be larger than the divisor? Why or why not?

## Concatenation

The + operator doesn’t just add two numbers, it can also “add” two strings! The process of combining two strings is called [*string concatenation*](https://www.codecademy.com/resources/docs/python/strings?page_ref=catalog). Performing string concatenation creates a brand new string comprised of the first string’s contents followed by the second string’s contents (without any added space in-between).

greeting\_text = "Hey there!"  
question\_text = "How are you doing?"  
full\_text = greeting\_text + question\_text  
   
# Prints "Hey there!How are you doing?"  
print(full\_text)

In this sample of code, we create two variables that hold strings and then concatenate them. But we notice that the result was missing a space between the two, let’s add the space in-between using the same concatenation operator!

full\_text = greeting\_text + " " + question\_text  
   
# Prints "Hey there! How are you doing?"  
print(full\_text)

Now the code prints the message we expected.

If you want to concatenate a string with a number you will need to make the number a string first, using the [str() function](https://www.codecademy.com/resources/docs/python/built-in-functions/str?page_ref=catalog). If you’re trying to print() a numeric variable you can use commas to pass it as a different argument rather than converting it to a string.

birthday\_string = "I am "  
age = 10  
birthday\_string\_2 = " years old today!"  
   
# Concatenating an integer with strings is possible if we turn the integer into a string first  
full\_birthday\_string = birthday\_string + str(age) + birthday\_string\_2  
   
# Prints "I am 10 years old today!"  
print(full\_birthday\_string)  
   
# If we just want to print an integer   
# we can pass a variable as an argument to   
# print() regardless of whether   
# it is a string.  
   
# This also prints "I am 10 years old today!"  
print(birthday\_string, age, birthday\_string\_2)

Using str() we can convert variables that are not strings to strings and then concatenate them. But we don’t need to convert a number to a string for it to be an argument to a print statement.

## Plus Equals

Python offers a shorthand for updating variables. When you have a number saved in a variable and want to add to the current value of the variable, you can use the += [(plus-equals) operator](https://www.codecademy.com/resources/docs/python/operators?page_req=catalog).

# First we have a variable with a number saved  
number\_of\_miles\_hiked = 12  
   
# Then we need to update that variable  
# Let's say we hike another two miles today  
number\_of\_miles\_hiked += 2  
   
# The new value is the old value  
# Plus the number after the plus-equals  
print(number\_of\_miles\_hiked)  
# Prints 14

Above, we keep a running count of the number of miles a person has gone hiking over time. Instead of recalculating from the start, we keep a grand total and update it when we’ve gone hiking further.

The plus-equals operator also can be used for string concatenation, like so:

hike\_caption = "What an amazing time to walk through nature!"  
   
# Almost forgot the hashtags!  
hike\_caption += " #nofilter"  
hike\_caption += " #blessed"

We create the social media caption for the photograph of nature we took on our hike, but then update the caption to include important social media tags we almost forgot.

## Multi-line Strings

Python strings are very flexible, but if we try to create a string that occupies multiple lines we find ourselves face-to-face with a SyntaxError. Python offers a solution: [*multi-line* strings](https://www.codecademy.com/resources/docs/python/strings?page_ref=catalog). By using three quote-marks (""" or ''') instead of one, we tell the program that the string doesn’t end until the next triple-quote. This method is useful if the string being defined contains a lot of quotation marks and we want to be sure we don’t close it prematurely.

leaves\_of\_grass = """  
Poets to come! orators, singers, musicians to come!  
Not to-day is to justify me and answer what I am for,  
But you, a new brood, native, athletic, continental, greater than  
  before known,  
Arouse! for you must justify me.  
"""

In the above example, we assign a famous poet’s words to a variable. Even though the quote contains multiple linebreaks, the code works!

If a multi-line string isn’t assigned a variable or used in an expression it is treated as a comment.

**Review**

In this lesson, we accomplished a lot of things! We instructed our computers to print messages, we stored these messages as variables, and we learned to update those messages depending on the part of the program we were in. We performed mathematical calculations and explored some of the mathematical expressions that Python offers us. We learned about errors and other valuable skills that will continue to serve us as we develop our programming skills.

<https://www.codecademy.com/resources/docs/python?page_ref=catalog>

<https://www.codecademy.com/workspaces/63ea8bc8206871a64de541fe>