# **Programming with Python**

# Module 03:



Programming languages all feature various programming statement and constructs. In this module you will learn some of the most common ones used in programming.

This modules will cover the following topics:

* **Using an Integrated Development Environment (IDE)**
* Pseudocode
* Conditional Statements
* Comparison Operators
* Loops
* Program Arguments
* Writing data to a File

## Using an Integrated Development Environment (IDE)

“An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion.” (Wikipedia, 2015, <https://en.wikipedia.org/wiki/Integrated_development_environment>)

“The free open-source edition of PyCharm,the premier IDE for pure Python development.” (JetBrains, 2015, <https://www.jetbrains.com/pycharm/>)

## For this module you will need to download and install PyCharm on your PC or Mac. You will find the instructions here:

## <https://www.jetbrains.com/pycharm/help/basics-and-installation.html>

## Pseudo Code



Pseudo Code is a fancy term for outlining your program's logic. In a text book or a document like this one, you just include it in the text as an example, but an example that will not necessarily run.

“Textbooks and [scientific publications](https://en.wikipedia.org/wiki/Scientific_publication) related to [computer science](https://en.wikipedia.org/wiki/Computer_science) and [numerical computation](https://en.wikipedia.org/wiki/Numerical_computation) often use pseudocode in description of algorithms, so that all programmers can understand them, even if they do not all know the same programming languages. In textbooks, there is usually an accompanying introduction explaining the particular conventions in use. The level of detail of the pseudo-code may in some cases approach that of formalized general-purpose languages.” (Wikipedia, 2015, <https://en.wikipedia.org/wiki/Pseudocode>)

In a programming script you usually write it in comments so that it can easily be converted into real programming statements, but it can also be written on a white board during a meeting.

# Get user data

# Print user data

Becomes

# Get user data

*strData = input("Enter your data: ")*

# Print user data

*print(strData)*

## Conditional Statements



There are two conditional statements that are included in most programming languages. The first is the “if-else” construct and the second is “switch-case.”

Python does not have a switch-case statement, but it does have something similar called "elif" clause which we will look at in just a bit.

### Using if-else

Using if-else is the most common way to create a conditional set of statements. The ***general pattern*** of an if-else statement is shown in the following pseudo code:

*if (condition1 == true)*

*{*

*// statements executed only if condition1 is true.*

*}*

*else*

*{*

*//statements executed only if condition1 is NOT true.*

*}*

Here is a simple Python example:

intVar = 1

**if** (intVar == 1):

print("1") # Statements executed only if condition1 is true.

**else**:

print("other")

**Note**: In Python, like many other languages, the single equals operator (=) is used to assign a value of 1 from the right side of the operator to the variable x on the left side of the operator. However, to compare the validity of the question “Does x equal 1” you use the equity operator of two equal signs (==).

Each if or else block can hold one or more statements.

intVar = 1

if (intVar == 1):

**print("Statement 1")**

**print("Statement 2")**

else:

print("another statement")

If there is only one statement you can place it on the same line as the if or else keywords.

intVar = 1

**if** (intVar == 1)**:** **print("Only 1 Statement")**

**else: print("another statement")**

### if - elif statements (Python’s Switch-Case option)

“In most languages, a switch statement is defined across many individual lines using one or two keywords. A typical syntax is:

* The first line contains the basic keyword, usually switch, case or select, followed by an expression which is often referred to as the control expression or control variable of the switch statement.
* Subsequent lines define the actual cases (the values) with corresponding sequences of statements that should be executed when a match occurs.”

(Wikipedia 2015, <https://en.wikipedia.org/wiki/Switch_statement>)

The ***general pattern*** of a Switch-Case statement is shown in the following pseudo code:

*switch(expression) {*

*case n:*

*code block*

*break; //Do not ‘fall through’ to check the next option*

*case n:*

*code block*

*break; //Do not ‘fall through’ to check the next option*

*default:*

*default code block*

*}*

“An [**if**](https://docs.python.org/2/reference/compound_stmts.html#if) ... [**elif**](https://docs.python.org/2/reference/compound_stmts.html#elif) ... [**elif**](https://docs.python.org/2/reference/compound_stmts.html#elif) ... **sequence** is a **substitute for the switch** or case statements found in other languages.” (Python Docs, 2015, <https://docs.python.org/2/tutorial/controlflow.html>)

Here is an example of a Python if-elif statement

intVar = 2

**if** (intVar == 1):

print("1")

**elif** (intVar == 2):

print("2")

**else**:

print("Please choose 1 or 2")

### Multiple Conditions

You can include more than one condition by using the operators or | and

intVar = 1

strName = "Bob"

if (intVar == 1 ***and*** strName == "Bob"):

print("1 and Bob")

else:

print("other")

intVar = 2

strName = "Bob"

if (intVar == 1 ***or*** strName == "Bob"):

print("1 and Bob")

else:

print("other")

### Nested if statements

There allow you to test multiple nested conditions.

intVar = 1

strName = "Bob"

**if** (intVar == 1):

print("1") # Statements executed only if condition1 is true.

**if**(strName == "Bob"):

print("Bob")

**else**:

print("other")



**LAB 03-1: If statements**

1. Type out and run the following code. Then explain what it does and does not do.

intVar = 2

if (intVar == 1):

print("1")

if (intVar == 2): # This is a nested if statement.

print("2")

else:

print("other")



## Comparison operators

“There are eight comparison operations in Python. They all have the same priority (which is higher than that of the Boolean operations). Comparisons can be chained arbitrarily; for example, x < y <= z is equivalent to x < y and y <= z, except that y is evaluated only once (but in both cases z is not evaluated at all when x < y is found to be false).

This table summarizes the comparison operations:” (Python doc, 2015, <https://docs.python.org/3.5/library/stdtypes.html>)

| **Operation** | **Meaning** |
| --- | --- |
| < | strictly less than |
| <= | less than or equal |
| > | strictly greater than |
| >= | greater than or equal |
| == | equal |
| != | not equal |
| is | object identity |
| is not | negated object identity |

### Comparing strings

Strings are case-sensitive, so you need to be somewhat careful when comparing them.

strName = "**b**ob"

if (strName == "**B**ob"): print("true")

else: print("**false**")

# It is common to covert data to a upper or lower case for comparisons

if (strName.**lower()** == "Bob".**lower()):** print("**true**")

else: print("false")

### The not operator

This operator reverses the results of a Boolean expression. The ***!*** is a common notation for ***not*** in most languages.

strVar = "Bob"

if (strVar == "Bob"): print("**true**")

else: print("false")

# not reverses true to false

strVar = "Bob"

if **not**(strVar == "Bob"): print("true")

else: print("**false**")

# The ! is a common symbol for not

if (strVar **!**= "Bob"): print("true")

else: print("**false**")

### Comparing Objects

If you make your own custom classes and objects in python the comparison is made on the *address in memory and not their content.*

class **Demo**: # This line of code defines a class named “Demo”

pass # This line tells Python that there is no code to process yet!

objD1 = **Demo**() # Make a new Demo object

objD2 = **Demo**() # Make another new Demo object

if (objD1 **==** objD2):

print("same")

else:

print("**different**")

objD1 **=** objD2 # Have the objD1 variable point to the objD2 **address**

if (objD1 **==** objD2):

print("**same**")

else:

print("different")

### Is operator

The operators “is”, and “is not”, also test for object identity. So (*x* ***is*** *y*) is true if and only if x and y are the same object in memory.

class Demo: # This line of code defines a class named “Demo”

pass # This line tells Python that there is no code to process yet!

objD1 = Demo() # Make a new demo object

objD2 = Demo() # Make another demo object

if (objD1 **is** objD2):

print("same")

else:

print("**different**")

objD1 = objD2 # Have the objD1 variable point to the objD2 **address**

if (objD1 **is** objD2):

print("**same**")

else:

print("different")

The *exception* to these is when you are comparing two object made from objects that act like “Value” types like strings, int, decimal, etc.

strName = "Bob"

strNickName = "Bob"

if (strName **==** strNickName): # Compares the **VALUE** not the address

print("**same**")

else:

print("different")

if (strName **is** strNickName): # Compares the **VALUE** not the address

print("**same**")

else:

print("different")

# Now, lets turn the string into a list

strName = **[**"Bob"**]**

strNickName = **[**"Bob"**]**

if (strName **==** strNickName): # **STILL**, compares the **VALUE** not the address

print("**same**")

else:

print("different")

if (strName is strNickName): # Compares the **ADDRESS** not the value!

print("Same")

else:

print("**different**")# Will be different this time!

NOTE: Just like the last time we talked about the differences between Value and Reference types, this may be confusing.

Remember that as long as you know that this behavior exists, and remember to test your code, it is just something you will get used to.

(If in doubt, test it out!)

## Boolean Values

In most languages zero evaluates, as the Boolean False while any other number evaluates are True. This is the same for empty strings (indicated by two quotation marks with no characters in between them). Let’s see what is true for Python!

print("using **Boolean** values")

if(**True**): print("**T**") # Will be true

else: print("F")

if(**False**): print("T")

else: print("**F**") # Will be false

print("Using **Numbers**")

if(**1**): print("**T**") # Will be true

else: print("F")

if(**0**): print("T")

else: print("**F**") # Will be false

print("using **Strings**")

if("**abc**"): print("**T**") # Will be true

else: print("F")

if(**""**): print("T") # Note I am using and empty string ""

else: print("**F**") # Will be false

Be careful, in python 0 is always equal to False, but only 1 is equal to True.

if(True == **1**): # Is True the same as 1?

print("**Yes**")

else:

print("No")

if(False == **0**): # Is False the same as 0?

print("**Yes**")

else:

print("No")

if(True == **2**): # Is True the same as 2?

print("Yes")

else:

print("**No**")

Here is a more complex example:

intX = 100

intY = 100

print("Zero test")

if( **0** == (**intX == intY**) ): # Does False == (True)

print("true")

else:

print("**false**") # Will be false

print("One test")

if( **1** == (**intX == intY**) ): # Does True == (True)

print("**true**") # Will be true

else:

print("false")

print("Neither Zero or One test")

if( **2** == (**intX == intY**) ): # However…

print("true")

else:

print("**false**")

print("Boolean True test")

if( **True** == (**intX == intY**) ): # …and how about this one?

print("**true**") #Will be True

else:

print("false")

### The keyword None

In Python, the keyword *None* is used to indicate that a value is not assigned a value. (Other languages use the word Null)

strData = **None**

if(strData):

print("T")

else:

print("**F**") # Will be false

## Loops



### while loop

"The while statement is used for repeated execution as long as an expression is true:" -- Python help files

intCounter = 0

**while**(**intCounter < 3**):

print(intCounter)

intCounter = intCounter + 1

Another common pattern is using a "Flag" value to stop the loop.

strUserInput = input("Type in a string to echo (Enter 0 to quit!)")

**while**(**strUserInput != "0"**): #Make sure to use Quotes!!

print(strUserInput)

strUserInput = input("Type in a string to echo (Enter 0 to quit!)")

### break

"The break statement, like in C, breaks out of the smallest enclosing *for* or *while* loop." -- Python help files

strUserInput = ""

**while**(**True**):

strUserInput = input("Type in a string to echo (Enter 0 to quit!)")

if(strUserInput == "0"): **break**

else: print(strUserInput)

### continue

"The continue statement, also borrowed from C, continues with the next iteration of the loop." -- Python help files

strUserInput = ""

while(True):

strUserInput = input("Type in a string to echo (Enter 0 to quit!)")

if(strUserInput == "0"): break

if(strUserInput == "C"): **continue**

print(strUserInput)

## Program Arguments



Perhaps a more useful example is one where you check to see if any arguments were passed in to a Console application and respond to what you find.

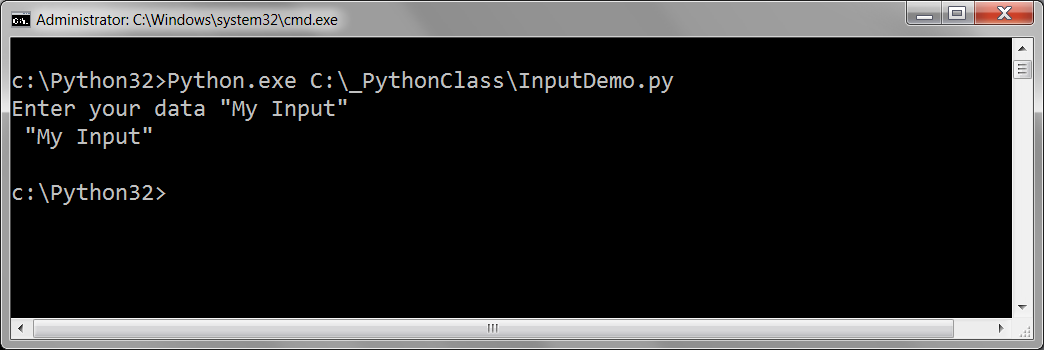
To understand arguments consider this example:

#Example 1: Pause and Ask for input

strData = **input**("Enter your data") # Get user input by pausing the program

print(strData) # Print user data

In Example 1, you run the script; it pauses, *waits for user input*, and then prints the data to the screen.



The next example is similar, except that it never pauses to ask the user for data:

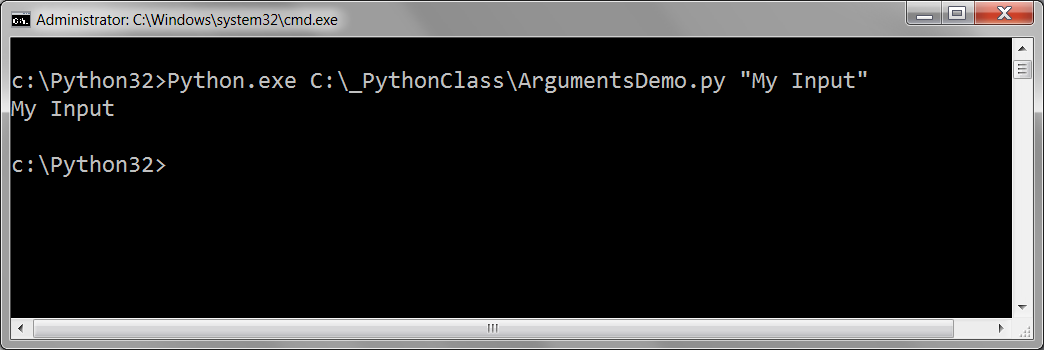
import sys #This forces your script to reference Python's *system* code module

#Example 2: Input is added when the script starts

strData = sys.**argv**[1] # Get user input as a script argument

print(strData) # Print user data

In Example 2, you still run the script, but you include some *user input at time it script starts*! These inputs are known as Arguments



**NOTE:** When using Windows, make sure you include **both** ***Python.exe*** and the script name to run the script with arguments.

When you run Console applications, you can add arguments to the command and they will be passed in as a collection of values. In Python, these values are store in a built-in the system variable called ***argv* (**which stands for "Argument Values").

The argv system variable can hold more than one value at a time, and you can access the individual values by indicating an id number, or "index."

The pattern is…

* The value of argv[0] will be the script name (and path on a PC).
* The value of argv[1] will be the first argument passed into the script.
* The value of argv[2] will be the second argument passed into the script after a space.
* The value of argv[n] will be the nth argument passed into the script after a space.

import sys

strData = sys.argv[0] # Get Script name

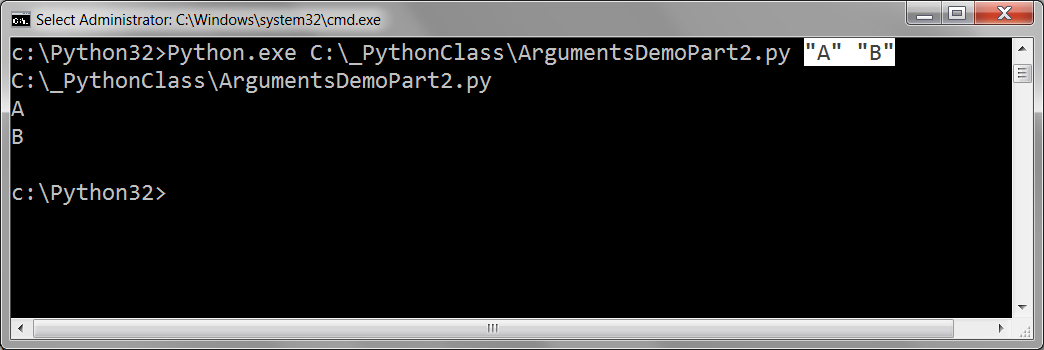
print(strData)#Print Script name

strData = sys.argv[1] # Get Argument 1

print(strData)#Print Argument 1 data

strData = sys.argv[2] # Get Argument 2

print(strData)#Print Argument 2 data



If there are no arguments then the expression “len(sys.argv) == 0” will evaluate to true, since the length function, len(), returns the number of items in the collection.

import sys

if(len(sys.argv) > 2):

intArg1 = int( sys.argv[1] ) # Get Argument 1

intArg2 = int( sys.argv[2] ) # Get Argument 2

strData = str(intArg1 + intArg2) # Perform some Processing

print("The Sum of the first and second arguments is: " + strData)

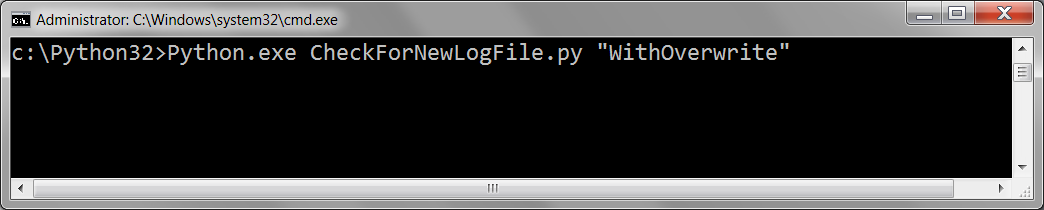
else:

print("This script requires two argument to run")

print("Ex: MyScript Arg1 Arg2")

Argument can be very handy to use with a Console applications, for it allows you to run scripts without additional user input. An example would be a script that would run at night using a scheduling program and no human interaction.

Let's say you had a program that copied several log files and placed them into a central folder. You could program an argument that asked if they wanted to overwrite existing files or just ignore them. This way the same script can be used in different ways.





**LAB 03-2: Script arguments**

1. Write a script that lets a user select one of three options by entering an argument when the script starts.
2. Start by typing these comments out and use them as pseudo code.

# Get the argument value

# Execute if 1, 2, or 3 is selected

# Print "You chose one" only if option 1 is selected.

# Print "You chose two" execute only if option 2 is selected.

# Print "You chose three" execute only if option 3 is selected.

# Print "Please choose 1, 2, or 3"

1. Now add if, elif, and else statements to your script so that it prints out an appropriate message based on the number passed to the script when it executes.



## Writing Data to a File



Writing data to files is a practical use of while loops and if statements. Here is an example of how to write to a file:

objFile = **open**("C:\\\_PythonClass\\TestData.txt", "a")

*# For more information on the open() function see page 193 of your book*

objFile.**write**(input("Enter your data: ") + "\n")

objFile.**close**()

Now we add a loop and conditional to the program.

objFile = **open**("C:\\\_PythonClass\\TestData.txt", "a")

print("Type in a string to write (Enter '*Exit*' to quit!)")

**while**(True):

strUserInput = input("Enter your data: ")

if(strUserInput.lower() == "exit"): **break**

else: objFile.**write**(strUserInput + "\n")

objFile.**close**()



**LAB 3-3: Working with Conditionals, Loops, and Files**

1. Create a script that lets a user add two numbers together and saves the answer to a file. Let the user continue adding numbers together until they type in the word "exit"

Hints:

* Use Pseudo code
* This time we are using the input() function and not script arguments



## Review Chapter 3

# “Chapter 3: Branching, While Loops, and Program Planning: The Guess My Number Game

## Overview

So far, the programs you’ve written have had a simple, sequential flow: each statement is executed once, in order, every time. If you were limited to just this type of programming, it would be very difficult, if not impossible, to write complex applications. But in this chapter, you learn how to selectively execute certain portions of your code and repeat parts of your program. Specifically, you learn to do the following:

* Generate random numbers using randint() and randrange()
* Use if statements to execute code based on a condition
* Use an else clause to make a choice based on a condition
* Use elif clauses to make a choice based on several conditions
* Use while loops to repeat parts of your program
* Plan your programs using pseudocode”

(Python Programming for the Absolute Beginner, Third Edition, by Michael Dawson)