Learning to Code with Python

# Module 1 Getting started

## How to install the tools to code in Python

The [installation steps](https://pytools.codeplex.com/wikipage?title=PTVS%20Installation) are explained at the Python Tools for Visual Studio website

1. Install [Visual Studio 2013 for desktop](http://www.microsoft.com/en-us/download/details.aspx?id=40787) (free)
2. Install [Visual Studio 2013 Update 2](http://www.microsoft.com/en-us/download/details.aspx?id=42666) so you have the latest features
3. Install [Python Tools for Visual Studio](https://pytools.codeplex.com/releases/view/119891)
4. Install the [Python 3.4 interpreter](https://www.python.org/ftp/python/3.4.1/python-3.4.1.msi)

## How do you create a Hello World project to test your installation?

Start Visual Studio

1. From the top menu select **File | New | Project**
2. Select **Installed | Templates | Python**
3. Select the project type: **Python Application**
4. Type in a name for your project: ***HelloWorld***
5. Select **OK**



After you select OK you will see the code editor window appear on the screen with a single line of code



That line of code is Python code that will print the words ‘Hello World’ on the screen

To run the program you can either use the function key F5, or you can select **Debug| Start Debugging** from the menu, or you can press the Start Debugging button in the toolbar.



When you run the program, if you see the following output, then you have successfully installed Visual Studio and the Python Tools for Visual Studio, and you are ready to code!

*NOTE: If you got an error message saying something about not being able to find the interpreter, then it is possible Visual Studio can’t find it. You can find instructions on how to tell Visual Studio how to locate the interpreter manually* [*here*](http://pytools.codeplex.com/wikipage?title=Selecting%20and%20Installing%20Python%20Interpreters)*. Just scroll down to the part of the page that says “Hey, I already have an interpreter on my machine, but PTVS doesn’t seem to know about it”. Follow the instructions then try to run your program again.*



To exit the program, you can press any key to continue as indicated in the displayed window, or you can use the function key SHIFT+F5, or you can select **Debug | Stop Debugging** from the menu, or you can press the Stop Debugging button in the toolbar.



You are now a programmer!

# Module 2 Displaying Text

## Print

Text is displayed on the screen using the print statement. The string to be displayed can be enclosed in single or double quotes.

**print("Hickory Dickory Dock! The mouse ran up the clock")**

**print('Hickory Dickory Dock! The mouse ran up the clock')**

## Displaying Text over multiple lines

You can use multiple print statements

**print('Hickory Dickory Dock!')**

**print('The mouse ran up the clock')**

You can use triple quotes

**print('''Hickory Dickory Dock!**

**The mouse ran up the clock''')**

You can use special characters to indicate a new line

**print('Hickory Dickory Dock!\nThe mouse ran up the clock')**

# Module 3 String Variables

The Input statement allows you to prompt a user for a value. You specify a variable to hold the value typed in by the user

**firstName = input(“What is your name?”)**

Once you have a value stored in a variable you can access it later in your code

**print(firstName)**

You can also change the value of the variable

**firstName = “Mike”**

There are some rules and guidelines for variable names

* Should be meaningful (e.g. firstName not variable1)
* Should be specific (birthDate not Date)
* Cannot contain spaces (firstName not first Name)
* Should be descriptive but not too long (favoriteSign not yourFavoriteSignInTheHoroscope)
* Are case sensitive (firstName and firstname would be two different variables)
* Cannot start with a number (name1 is okay 1name is not)

You can concatenate variables and strings together using the “+” symbol

**print(firstName + lastName)**

**print("Hello " + firstName + " " + lastName)**

When strings are stored in variables, you can use string functions to manipulate the contents of the variable or to act on the contents of the variable

|  |  |
| --- | --- |
| **firstname.lower()** | Returns string in lowercase |
| **firstname.upper()** | Returns the string in uppercase |
| **firstname.swapcase()** | Returns lowercase letters in uppercase and uppercase letters in lowercase |
| **firstname.capitalize()** | Returns the string with the first letter capitalized |
| **firstname.find(“X”)** | Returns the position of “X” in the string |
| **firstname.replace(“X”,”Y”)** | Replaces occurrences of “X” in the string with “Y” |
| **firstname.count(“X”)** | Tells you how many times “X” appears in the string |

For a complete list of string functions you can use with variables containing string visit

<https://docs.python.org/2/library/string.html>

# Module 4 Numeric variables

Variables can be used to store numbers as well as strings

**length = 42**

You can perform mathematical operations with numeric variables

**width = 20**

**area = length \* width**

Here are some common mathematical operations used in code

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Operation** | **Example** |
| + | Addition | 5+2=7 |
| - | Subtraction | 5-2=3 |
| \* | Multiplication | 5\*2=10 |
| / | Division | 5/2 = 2.5 |
| \*\* | Exponent | 5\*\*2=25 |
| % | Modulo | 5%2 = 1 |

It is important to remember the order of operations

() parentheses

\*\* exponent

\*/ multiplication and division

+ - addition and subtraction

In addition to math symbols, there are math functions you can use to operate on numeric values

<https://docs.python.org/2/library/math.html>

### How to format numeric output

|  |  |  |
| --- | --- | --- |
| % syntax | Format syntax | Output |
| **print(‘I have %d cats’% 6)** | **print(“I have {0:d} cats”.format(6)** | I have 6 cats |
| **print(‘I have %3d cats’ %6)** | **print ("I have {0:3d} cats".format(6))** | I have 6 cats |
| **print(‘I have %03d cats’ %6)** | **print("I have {0:03d} cats".format(6))** | I have 006 cats |
| **print(‘I have %f cats’ %6)** | **print("I have {0:f} cats".format(6))** | I have 6.000000 cats |
| **print(‘I have %.2f cats’ %6)** | **print("I have {0:.2f} cats".format(6))** | I have 6.00 cats |

Here is a nice summary of the % syntax for formatting

<http://infohost.nmt.edu/tcc/help/pubs/lang/pytut/str-format.html>

Here is a summary of the format syntax (it takes a little longer to master)

<http://legacy.python.org/dev/peps/pep-3101/>

### Datatype Conversions

|  |  |
| --- | --- |
| **int(x)** | Converts x to an integer |
| **long(x)** | Converts x to a long integer |
| **float(x)** | Converts x to a float |
| **str(x)** | Converts x to a string |

The complete list of functions for converting to different dataypes and many other built in functions you can use are listed here

<https://docs.python.org/2/library/functions.html>

# Chapter 5 Date Variables

The datetime module contains a function for retrieving the current date and time

**import datetime**

**currentDate = datetime.date.today()**

Accessing different parts of the date

**currentDate.year**

**currentDate.month**

**currentDate.day**

You can find a complete list of the functions in the datetime module here

<https://docs.python.org/2/library/datetime.html>

Formatting dates for display to users

currentDate.strftime('%d %b,%Y')

* %d is the day of the month
* %b is the abbreviation for the month
* %Y is the 4 digit year
* %B is the full month name
* %y is two digit year
* %a is the day of the week abbreviated
* %A is the day of the week

You can find a complete list of abbreviations to use with strftime here

<http://strftime.org/>

Converting strings containing dates into datetime variables

**birthday = input ("What is your birthday? ")**

**birthdate = datetime.datetime.strptime(birthday,"%m/%d/%Y").date()**

For more useful date functions check out the dateutil module

<http://labix.org/python-dateutil>

Working with times

The same function that returns current date also returns the time

**import datetime**

**currentTime = datetime.datetime.now()**

**print (currentTime)**

**print (currentTime.hour)**

**print (currentTime.minute)**

**print (currentTime.second)**

**datetime.datetime.strftime(currentTime,'%H:%M')**

Formatting options for times

**%H Hours (24 hr clock)**

**%I Hours (12 hr clock)**

**%p AM or PM**

**%m Minutes**

**%S Seconds**

# Module 5 Making Decisions with code

If statements allow you to specify code to execute only if a particular condition is true

if answer == "yes" **:**

    print("That will be an extra $10")

|  |  |  |
| --- | --- | --- |
| Symbol | Meaning | Example |
| == | Is equal to | if answer == “yes” : |
| != | Is not equal to | if answer != “no” : |
| < | Is less than | if total < 100 : |
| > | Is greater than | if total > 100 |
| <= | Is less than or equal to | if total <= 100 |
| >= | Is greater than or equal to | if total >=100 |

You can use Boolean variables to store a True or False value

**if float(deposit) > 100 :**

**#Set the boolean variable freeToaster to True**

**freeToaster=True**

**#if the variable freeToaster is True**

**#the print statement will execute**

**if freeToaster :**

**print("enjoy your toaster")**

You can find the documentation of the if statement here

<https://docs.python.org/2/tutorial/controlflow.html>

# Module 7 Complex decisions with code

You can check multiple conditions using an elif in your if statement

**if country == "CANADA" :**

**print("Hello")**

**elif country == "GERMANY" :**

**print("Guten Tag")**

**elif country == "FRANCE" :**

**print("Bonjour")**

You can add an else statement which will be executed if none of the conditions listed are true.

**if country == "CANADA" :**

**print("Hello")**

**elif country == "GERMANY" :**

**print("Guten Tag")**

**elif country == "FRANCE" :**

**print("Bonjour")**

**else :**

**print("Aloha/Ciao/G’Day")**

The AND statement allows you to specify multiple conditions that must ALL be true.

**if wonLottery and bigWin :**

**print("you can retire")**

|  |  |  |
| --- | --- | --- |
| First Condition is | Second Condition is | Statement is |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

The OR statement allows you to specify that only one condition must be true

**if saturday or sunday :**

**print("you can sleep in")**

|  |  |  |
| --- | --- | --- |
| First Condition is | Second Condition is | Statement is |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

You can combine multiple AND/OR in a single if statement

**if month == "Sep" or month =="Apr" \**

**or month == "Jun" or month == "Nov" :**

**print("There are 30 days in this month")**

**if favMovie == "Star Wars" \**

**and favBook == "Lord of the Rings" \**

**and favEvent == "ComiCon" :**

**print("You and I should hang out")**

You can combine AND/OR in a single statement

NOTE: AND statements are executed before OR statements, so you may need to add parentheses to ensure your statements are interpreted correctly and you do not get unexpected results.

**if country == "CANADA" and \**

**(pet == "MOOSE" or  pet == "BEAVER") :**

**print("Do you play hockey too")**

If statements can also be nested one inside another

**if monday :**

**#you could have code here to check for fresh coffee**

**# the if statement is nested, so this if statement**

**# is only executed if the other if statement is true**

**if not freshCoffee :**

**print("go buy a coffee!")**

**print("I hate Mondays")**

**print("now you can start work")**

# Module 8 Repeating Events

The turtle module provides methods you can use to draw lines and shapes

import turtle

turtle.color('green')

turtle.forward(100)

turtle.right(45)

Turtle commands

|  |  |
| --- | --- |
| Command | Action |
| **right(x)** | Rotate right x degrees |
| **left(x)** | Rotate left x defrees |
| **color(‘x’)** | Change pen color to x |
| **forward(x)** | Move forward x |
| **backward(x)** | Move backward x |

The for loop allows you to repeat a set of commands a specific number of times

**import turtle**

**for steps in range(4):**

**turtle.forward(100)**

**turtle.right(90)**

You can also nest loops inside each other

**import turtle**

**for steps in range(4):**

**turtle.forward(100)**

**turtle.right(90)**

**for moresteps in range(4):**

**turtle.forward(50)**

**turtle.right(90)**

You can specify the start and end values to use for the range

**for steps in range(1,4) :**

**print(steps)**

You can also specify a step, so your loop skips values

**for steps in range(1,10,2) :**

**print(steps)**

Python also allows you to explicitly list the values to use inside a loop

**for steps in [1,2,3,4,5] :**

**print(steps)**

**import turtle**

**for steps in ['red','blue','green','black'] :**

**turtle.color(steps)**

**turtle.forward(100)**

**turtle.right(90)**

Here is a good tutorial on for loops

<https://www.udemy.com/blog/python-for-loop/>

# Module 9 Repeating events until done

While loops execute over and over until the condition specified is no longer true

**import turtle**

**counter = 0**

**while counter < 4:**

**turtle.forward(100)**

**turtle.right(90)**

**counter = counter+1**

Be careful that something inside the loop will change the condition so you don’t end up with an endless loop

**import turtle**

**counter = 0**

**while counter < 3:**

**turtle.forward(100)**

**turtle.right(90)**

You can find more documentation on the turtle module here

<https://docs.python.org/2/library/turtle.html>

You can find more documentation on the while loop here

<http://www.tutorialspoint.com/python/python_while_loop.htm>

# Module 10 Remembering lists

Lists allow you to store multiple values

**scores = [78,85,62,49,98]**

**guests = ['Christopher','Susan','Bill','Satya']**

You can create an empty list and add values later

**guests = []**

**scores = []**

You can reference any value in the list by specifying it’s position

**#Print the fourth score**

**print(scores[3])**

**guests = ['Christopher','Susan','Bill','Satya']**

**#print the first guest**

**#the first value is in position 0**

**print(guests[0])**

**#print the last entry in the list**

**print(guests[-1])**

**#print the second last entry in the list**

**print(guests[-2])**

**scores = [78,85,62,49,98]**

You can change a value in a list

**guests[0] = 'Steve'**

You can add a value to a list with the append() function

**guests.append('Steve')**

You can remove a value from a list with the remove() function or the del command

**guests.remove('Christopher')**

**del guests[0]**

The index() function will search a list and return the index position where the value was found

**guests.index('Bill')**

If you search for a value that does not exist, the code will return an error (error handling is covered later)

If you know how many rows are in a list you can use a for loop to go through all the values in a list

**for steps in range(4) :**

**#Remember the value of steps goes up by one**

**#Each time the loop executes**

**print(guests[steps])**

If you don’t know how many values are in the list you can use the len() function to find out the number of entries in the list

**for steps in range(nbrEntries) :**

**print(guests[steps])**

There is also a specialized for loop that is designed to go through all the entries in a list

**guests = ['Christopher','Susan','Bill','Satya']**

**#specify the name of your list and a variable name**

**#to hold each entry as you go through the loop**

**for guest in guests :**

**#the variable guest will contain the values**

**#as we go through the loop**

**print(guest)**

You can sort a list using the sort() function

**guests.sort()**

You can find more information on lists here

<http://www.diveintopython.net/native_data_types/lists.html>

# Module 11 How to save information in files

Use the open() function to create a new file or open an existing file

**myFile = open(fileName, accessMode)**

The access mode specifies the action you want to do with the file after you open it.

r read the file

w write to the file

a Append to the existing file content

b open a binary file

**fileName = "GuestList.txt"**

**accessMode = "w"**

**myFile = open(fileName, accessMode)**

Then use the write function to write content to the open file

**myFile.write("Hi there!")**

If you want to start a new line, use the /n like we did with the print statement

**myFile.write("Hi there!\n")**

When you are done close the file

**myFile.close()**

You can find more information on how to write to files here

<https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files>

# Module 12 Reading from files

Use the open function to open the file and specify a suitable access mode

**myFile = open(fileName, accessMode)**

The access mode specifies the action you want to do with the file after you open it.

r read the file

w write to the file

a Append to the existing file content

b open a binary file

The read() function will read the entire file contents into a string variable

**fileContent= myFile.read()**

The readline() function will read one line from the file

**fileContent= myFile.readline()**

If you are reading a comma separated variable file there is a csv module you can use

**import csv**

The reader() function in the csv module will return all the rows from a csv file into a list

**dataFromFile = csv.reader(myCSVfile)**

If your file is not using a comma to separate the values, you can tell the reader() function which characters is used as a delimiter

**dataFromFile = csv.reader(myCSVFile, delimiter=",")**

The following code would open and read a csv file

**fileName = "GuestList.txt“**

**accessMode = "r"**

**with open(fileName, accessMode) as myCSVFile:      #Read the file contents**

**dataFromFile = csv.reader(myCSVFile)**

The with command ensures that if an error occurs in the code the file will still be closed

Once you have used the reader() function to get the file contents in a list you can use a for loop to access the contents of the list

**with open(fileName, accessMode) as myCSVFile:**

**#Read the file contents**

**dataFromFile = csv.reader(myCSVFile)**

**#For loop that will run once per row**

**for row in dataFromFile :**

**print(row)**

The approach above displays the output with square brackets. If you prefer there is a join function you can use to display the output. The syntax takes a little getting used to.

‘delimiter to display between values’.join(row)

For example in the above code you could replace the for loop with the following to show the rows with commas separating the values in the row

**for row in dataFromFile :**

**print (', '.join(row))**

If you want to access an individual value in a row instead of all the values in a row, you can use a nested loop

**for row in dataFromFile :**

**print(row)**

**for value in row :**

**print(value + "\n**")

You can find more information on how to read from files here

<https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files>

# Module 13 Functions

Functions are a callable section of code that allows you to avoid rewriting the same code over and over

Use the def keyword to indicate code that is contained in a function

**def printMessage():**

**print('Hello World')**

**return**

Call the function using the function name in your code

**printMessage()**

Functions can also accept parameters

**def printMessage(message):**

**print(message)**

**return**

**printMessage('Hello world!')**

Functions can return values

**def getMessage(name):**

**message = 'Hello, ' + name**

**return message**

**def printMessage(message):**

**print(message)**

**return**

**output = getMessage('Christopher')**

**printMessage(output)**

You can find more information on functions here

<http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/functions.html>

# Module 14 Handling Errors

If you have a line of code that may generate an error at runtime. You can use the try except to specify code to execute when an error occurs

**first = input("Enter the first number ")**

**second = input("Enter the second number ")**

**firstNumber = float(first)**

**secondNumber = float(second)**

**try :**

**result = firstNumber / secondNumber**

**print (first + " / " + second + " = " + str(result))**

**except :**

**print("I am sorry something went wrong")**

The sys.exc\_info() function allows you to find out the error message that you trapped in yoru code

**import sys**

**first = input("Enter the first number ")**

**second = input("Enter the second number ")**

**firstNumber = float(first)**

**secondNumber = float(second)**

**try :**

**result = firstNumber / secondNumber**

**print (first + " / " + second + " = " + str(result))**

**except :**

**error = sys.exc\_info()[0]**

**print("I am sorry something went wrong")**

**print(error)**

Once you know the specific error that occurred you can write code to handle that error specifically

**first = input("Enter the first number ")**

**second = input("Enter the second number ")**

**firstNumber = float(first)**

**secondNumber = float(second)**

**try :**

**result = firstNumber / secondNumber**

**print (first + " / " + second + " = " + str(result))**

**except ZeroDivisionError :**

**print("The answer is infinity")**

Typically you will trap any specific errors you can predict and then add a generic error handler for any other errors that may occur.

**first = input("Enter the first number ")**

**second = input("Enter the second number ")**

**firstNumber = float(first)**

**secondNumber = float(second)**

**try :**

**result = firstNumber / secondNumber**

**print (first + " / " + second + " = " + str(result))**

**except ZeroDivisionError :**

**print("The answer is infinity")**

**except :**

**error = sys.exc\_info()[0]**

**print("I am sorry something went wrong")**

**print(error)**

For a complete list of specific errors you can trap you can see a list here

<http://www.tutorialspoint.com/python/standard_exceptions.htm>

You can find additional help on how to trap errors here

<http://www.tutorialspoint.com/python/python_exceptions.htm>