Introduction to Programming using Python

Setting Expectations

- Target Audience
  - People new to programming
  - Students
  - Career changers
  - IT Pros
  - Anyone with an interest in learning to code
- If you want to follow along...
  - Install Visual Studio Express
  - Install the Python tools
    - Instructions coming soon...

Getting started

Why and How

Why Python?

- There are a LOT of different programming languages out there
- Python is one of the easier ones to learn
- There are lots of free tools out there you can use to code or learn Python
- There are a lot of different ways to use Python code

Python Overview

- Python is a high-level, interpreted, interactive and object oriented-scripting language.
- Python was designed to be highly readable which uses English keywords frequently where as other languages use punctuation and it has fewer syntactical constructions than other languages.
- Python is Interpreted: This means that it is processed at runtime by the interpreter and you do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive: This means that you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: This means that Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is Beginner's Language: Python is a great language for the beginner programmers and supports the development of a wide range of applications, from simple text processing to WWW browsers to games.

What is a Software Program?

Software Program
A software program is commonly defined as a set of instructions, or a set of modules or procedures, that allow for a certain type of computer operation.

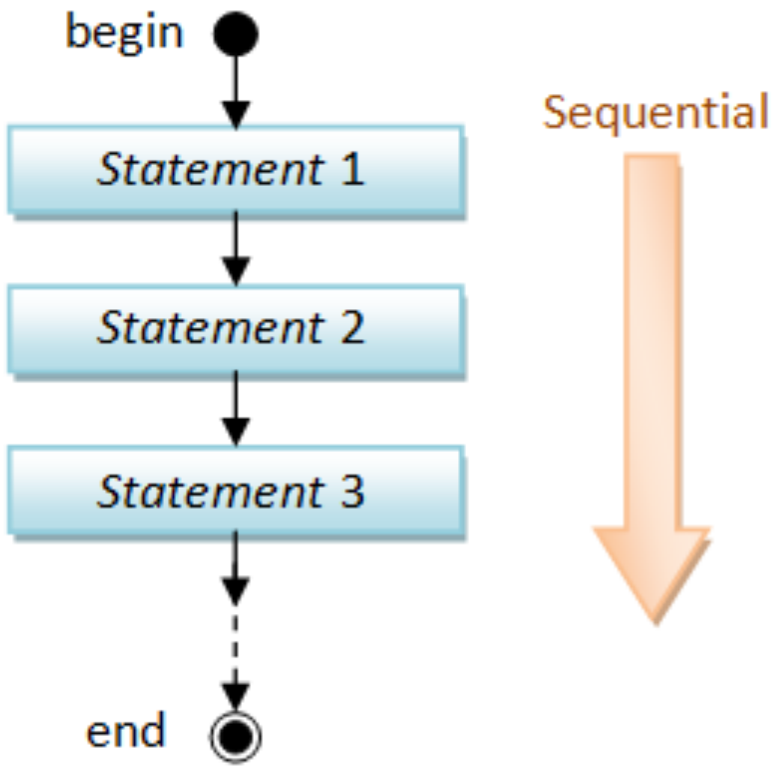
sequential flow
A program is a sequence of instructions (called programming statements), executing one after another in a predictable manner.

Sequential flow is the most common and straight-forward, where programming statements are executed in the order that they are written - from top to bottom in a sequential manner, as illustrated in the following flow chart.

The following program prints the area and circumference of a circle, given its radius.

Take note that the programming statements are executed sequentially - one after another in the order that they were written.

Sequential Execution



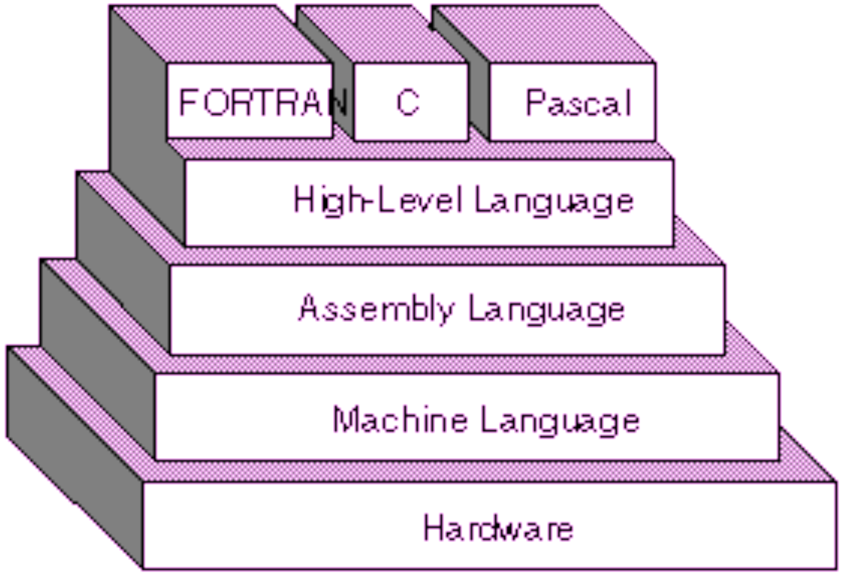
What is Programming Language?

A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

The term programming language usually refers to high-level languages, such as Ada, BASIC, C, C++, C#, COBOL, Java, FORTRAN, Groovy, LISP, Pascal, PHP, Python, Ruby, Scala, SmallTalk etc.

Each programming language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

What is Programming Language?



What is Assembley Language?

An assembly language is a low-level programming language for microprocessors and other programmable devices. It is not just a single language, but rather a group of languages.

An assembly language implements a symbolic representation of the machine code needed to program for a given CPU architecture.

Assembly language is also known as assembly code. The term is often also used synonymously with 2GL.

An assembly language is the most basic programming language available for any processor.

With assembly language, a programmer works only with operations that are implemented directly on the physical CPU.

Assembly languages generally lack high-level conveniences such as variables and functions, and they are not portable between various families of processors.

They have the same structures and set of commands as machine language, but allow a programmer to use names instead of numbers.
This language is still useful for programmers when speed is necessary or when they need to carry out an operation that is not possible in high-level languages.

What is Machine Language?

Machine language is the lowest-level programming language (except for computers that utilize programmable microcode).
Machine languages are the only languages understood by computers.

Why Humans Don't Use Machine Language?
While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.
Programmers, therefore, use either a high-level programming language or an assembly language.

An assembly language contains the same instructions as a machine language, but the instructions and variables have names instead of being just numbers

Programs written in high-level languages are translated into assembly language or machine language by a compiler.

Assembly language programs are translated into machine language by a program called an assembler.

Every CPU has its own unique machine language. Programs must be rewritten or recompiled, therefore, to run on different types of computers.

What is High Level Language?

A high-level language is a programming language designed to simplify computer programming.

It is "high-level" since it’s several steps removed from the actual code run on a computer's processor.

High-level source code contains easy-to-read syntax that is later converted into a low-level language, which can be recognized and run by a specific CPU.

Most common programming languages are considered high-level languages.
C++, C#, Cobol, Fortran, Java, JavaScript, Lisp, Objective C,
Pascal, Perl, PHP, Python, Scala, Swift etc.

Each of these languages use different syntax. Some are designed for writing desktop software programs, while others are best-suited for web development. But they all are considered high-level since they must be processed by a compiler or interpreter before the code is executed.

Source code written in languages like C++ and C# must be compiled into machine code in order to run.
The compilation process converts the human-readble syntax of the high-level language into low-level code for a specific processor.
Source code written in scripting languages like Perl and PHP can be run through an interpreter, which converts the high-level code into a low-level language on-the-fly.

What is a Compiler?

A compiler is a software program that transforms high-level source code that is written by a developer in a high-level programming language into a low level object code (binary code) in machine language, which can be understood by the processor.

The process of converting high-level programming into machine language is known as compilation.

The processor executes object code, which indicates when binary high and low signals are required in the arithmetic logic unit of the processor.

A compiler that converts machine language into high-level natural language is called a de-compiler.

Compilers that produce the object code meant to run on a system are called cross-compilers.

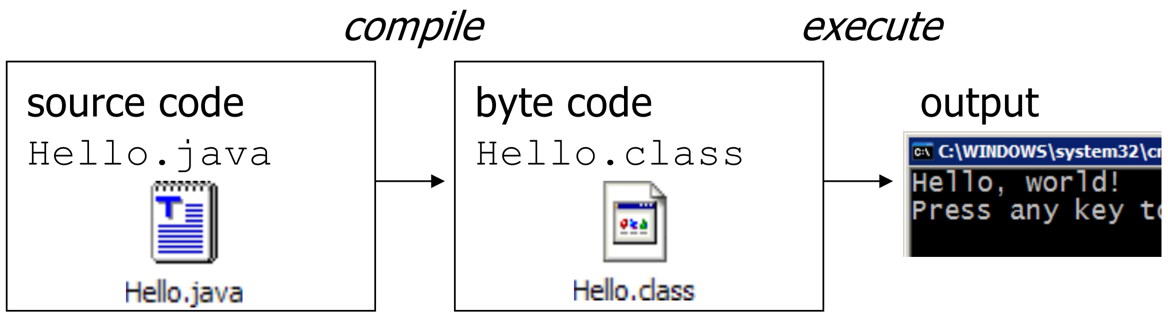
What is a Interpreter?

An interpreter is a computer program that is used to directly execute program instructions written using one of the many high-level programming languages.

The interpreter transforms the high-level program into an intermediate language that it then executes, or it could parse the high-level source code and then performs the commands directly, which is done line by line or statement by statement.

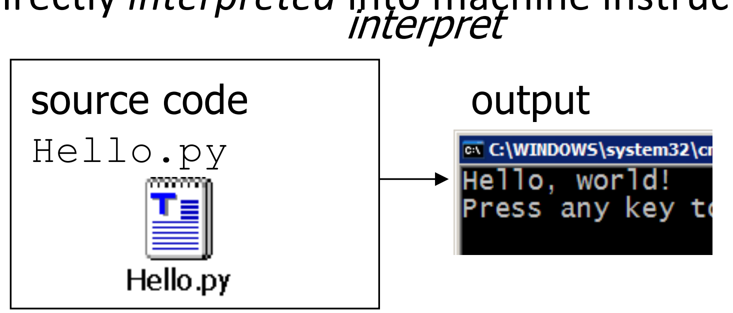
Compiling and interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



```
Hello, world!
Press any key t
```

- Python is instead directly *interpreted* into machine instructions.



```
Hello, world!
Press any key t
```

History of Python:

- Python 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 ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted, Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing it's progress.

Python Features

- Easy-to-learn: Python has relatively few keywords, simple structure, and a clearly defined syntax.
- Easy-to-read: Python code is much more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's success is that its source code is fairly easy-to-maintain.
- A broad standard library: One of Python's greatest strengths is the bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode: Support for an interactive mode in which you can enter results from a terminal right to the language, allowing interactive testing and debugging of snippets of code.

Python Features (cont'd)

- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable: Python provides a better structure and support for large programs than shell scripting.

Python Environment

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX etc.)
- Windows
- Apple Max
- OS/2
- DOS (multiple versions)
- PalmOS
- Nokia mobile phones
- Windows CE
- VMS/OpenVMS
- Python has also been ported to the Java and .NET virtual machines.

And as a bonus

- Once you learn how to code in one programming language it will be easier to learn another programming language, and another, and another…

JavaScript

???

C#

C++

Perl

But let’s be clear about something…

- You won’t learn enough in this course to start adding special effects to the next big superhero movie
- You WILL learn enough to start solving real world problems with code
- OR to just start having some fun ☺

So how do I get started?

You need to install software on your PC/laptop

- There are a lot of different tools out there you can use to write Python Code.
- In this course we will use Anaconda Spyder + Jupyter notebook + VSCode

http://www.cdt-pv.org/media/resources/Anaconda-Quickstart.pdf

Anaconda Configuration

```
#Update conda
conda update conda

#Update Jupyter
conda update jupyter

#Anaconda help
conda –h

#install packages
conda install numpy
conda install pandas
conda install statsmodels
conda install matplotlib
conda install seaborn

#List installed packages
conda list

#install non conda packages
pip install <package>
```

```
#To Open Jupyter notebook from Anaconda Prompt, type
jupyter notebook

#To Open Spyder from Anaconda Prompt, type
start spyder

# Jupyter notebook can run R Codes also
conda install -c r r-essentials
```

To create an environment with a python version:

```
conda create -n myenv python=3.7
```

Create the environment from the environment.yml file:

```
conda env create -f environment.yml
```

You can make an exact copy of an environment by creating a clone of it:

```
conda create --name myclone --clone myenv
```

To verify that the copy was made:

```
conda info --envs
```

Geek Tip!

- There are actually a lot of different version flavors of Python:
- Python 2.x and 3.x
- IronPython, IPython, CPython, PyPy, Jython, Canopy, Anaconda, …
- We will be using the Anaconda interpreter with Python 3.7
- So, if you copy code from a website and it doesn’t work don’t panic! It might just be a slightly different version of Python

How do I know I installed everything correctly?

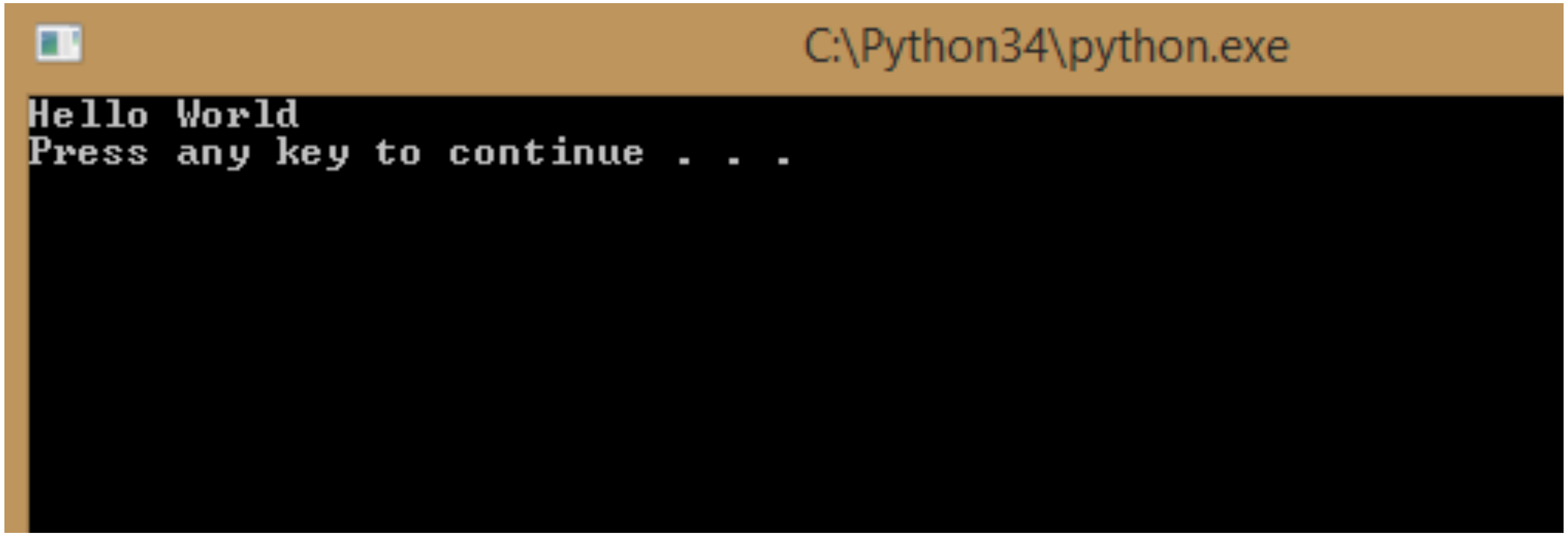
- There is a tradition among programmers
- We always test our installation by writing the same program:

DEMO

Creating your Hello World program!

You have now created your first application

```python
print('Hello World')
```



```
Hello World
Press any key to continue . . .
```

Best practices

Pick up good habits right away!

- Comments in your code help you or someone else understand
  - What your program does
  - What a particular line or section of code does
  - Why you chose to do something a particular way
  - Anything that might be helpful to know if I am looking at the code later and trying to understand it!

In Python we use a # to indicate comments

```python
#My first Python Application
#Created by me!
#Print command displays a message on the screen
print('Hello World')
```

Git & GitHub

What is Git and GitHub

Git

version control system
SVN,StarTeam

Github

repository site, there are others such as bitbucket and visual Studio online

Git Config

- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

- Let’s define ourselves first
  - git config --global user.email "you@example.com"
  - git config --global user.name "Your Name"

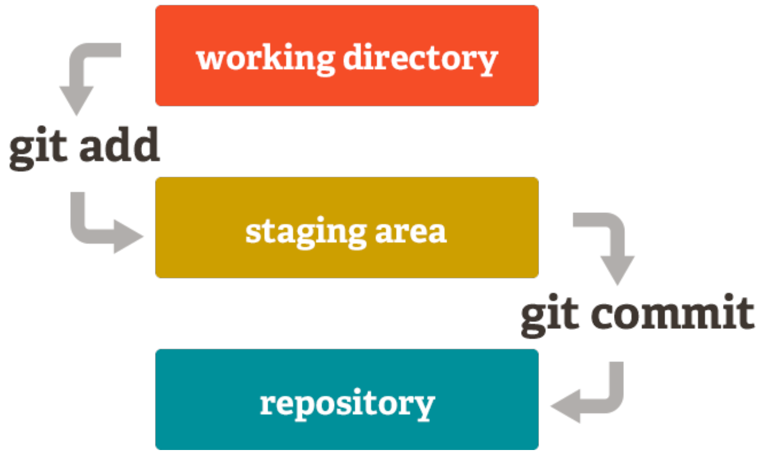
Local first

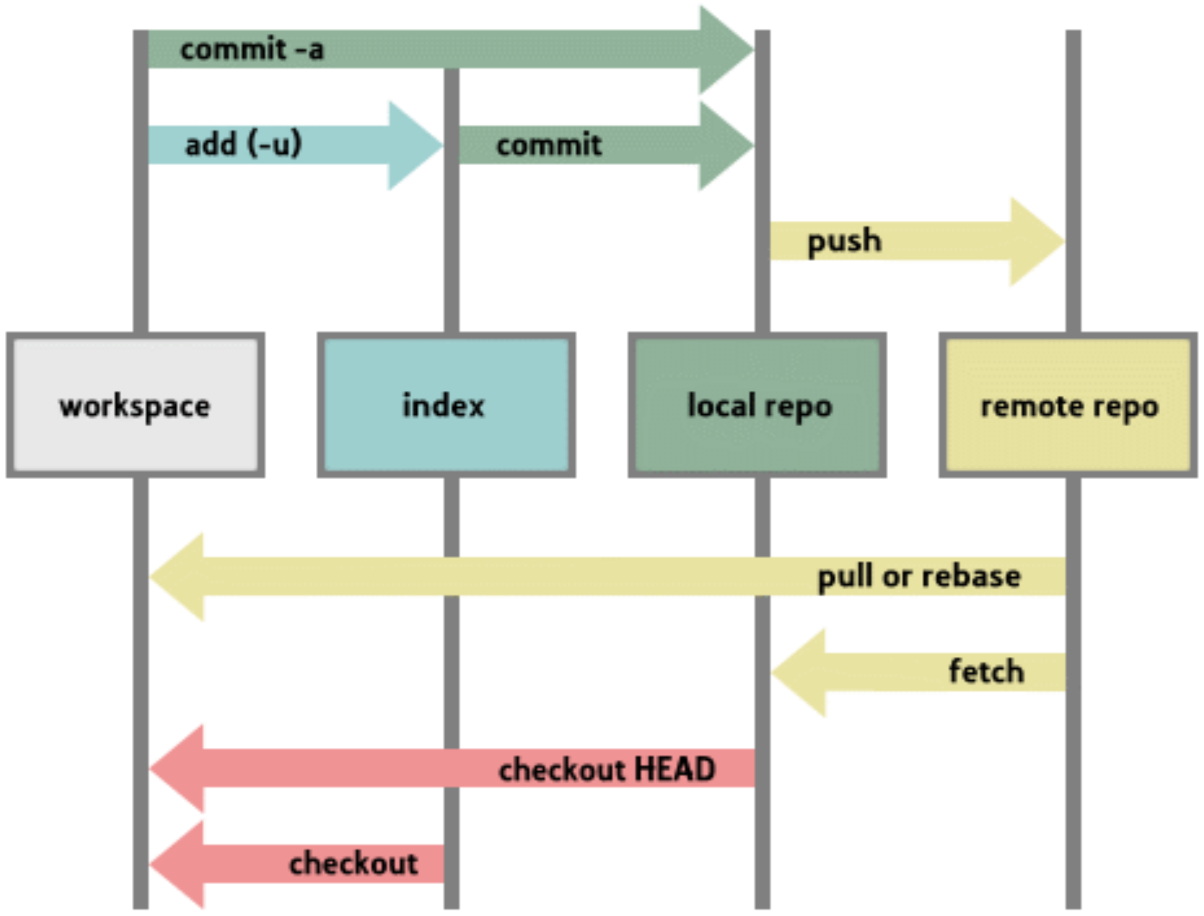
- first lets make a git folder in our computer
  - git init
- add a file to the folder
  - git add newfile.file
  - git commit -m “new file added” newfile.file
  - If you don’t provide file after the comment, everything you have done will be committed.

Ignore?

- Create a file called .gitignore
  - Write the file names that should be ignored by git
  - Commit the file

Git Indexing





Pulling a repository/editing

- lets pull a repository from github
  - git pull https://github.com/ajith-kl2018/pythonmay20
  - git remote add origin https://github.com/ajith-kl2018/pythonmay20
  - defined the link as origin. No need to write this link every time
- For Cloning an existing remote repo
  - git clone https://github.com/ajith-kl2018/pythonmay20
- Edit the file a.txt
- git add .
- git commit –am ‘something added’
- git push origin

Git Pull / Download my Code

- git clone https://github.com/ajith-kl2018/pythonmay20
- cd pythonmay20
- git pull origin

Remove a file

- git rm somefile.txt
- git commit –m ‘removed’
- git push origin

Conflict

- git commit -am 'Conflicts resolved‘

Go back to older version

- Git log
  - Show me the logs git☺
- See the commit id
  - git reset --hard c1ac571d9af5fe1126ecdf64c57d6ef4a5990a60 && git clean -f
- Destroys the local modifications!
- Removes untracked files!

One step back!

- An easy way to revert last commit (1)
  - git revert HEAD~1..HEAD
  - git push origin

Displaying text print

Many computer programs provide information

- One of the simplest but important things you need the ability to do in your code is display text

The print statement is used to display text

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

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

You can use single quotes or double quotes

DEMO

Printing text

Multiple lines

Does it matter if you use single or double quotes?

```
print("It's a beautiful day in the neighborhood")

print('It's a beautiful day in the neighborhood')
```

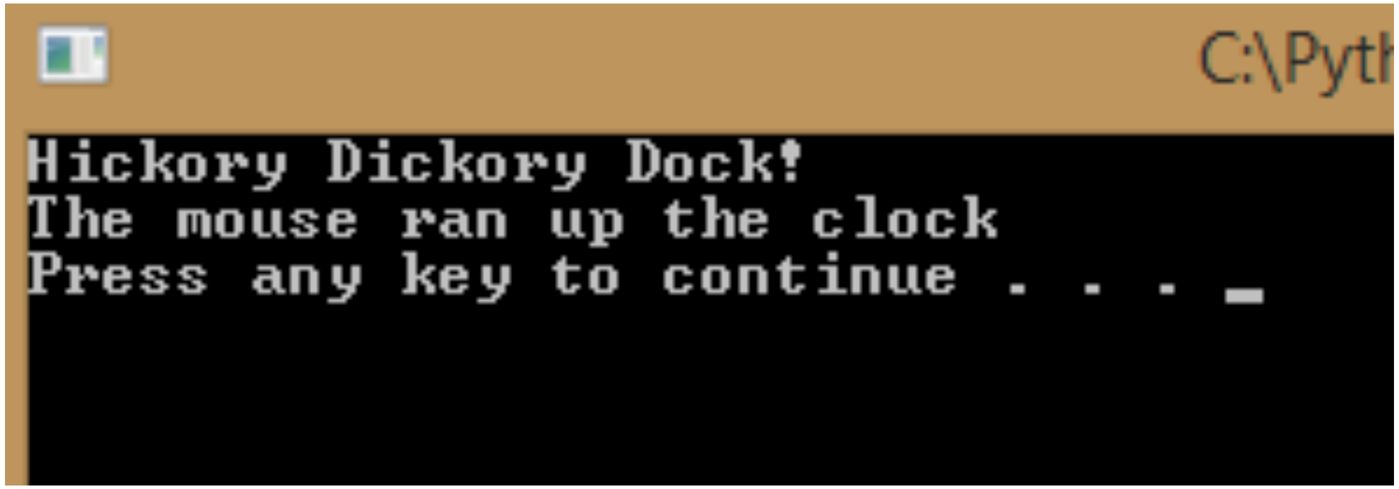
Only if the string you are displaying contains a single or double quote.

It’s a good habit to pick one and stick with it as much as possible.

What if I want my text to appear on multiple lines?

You can use multiple print statements

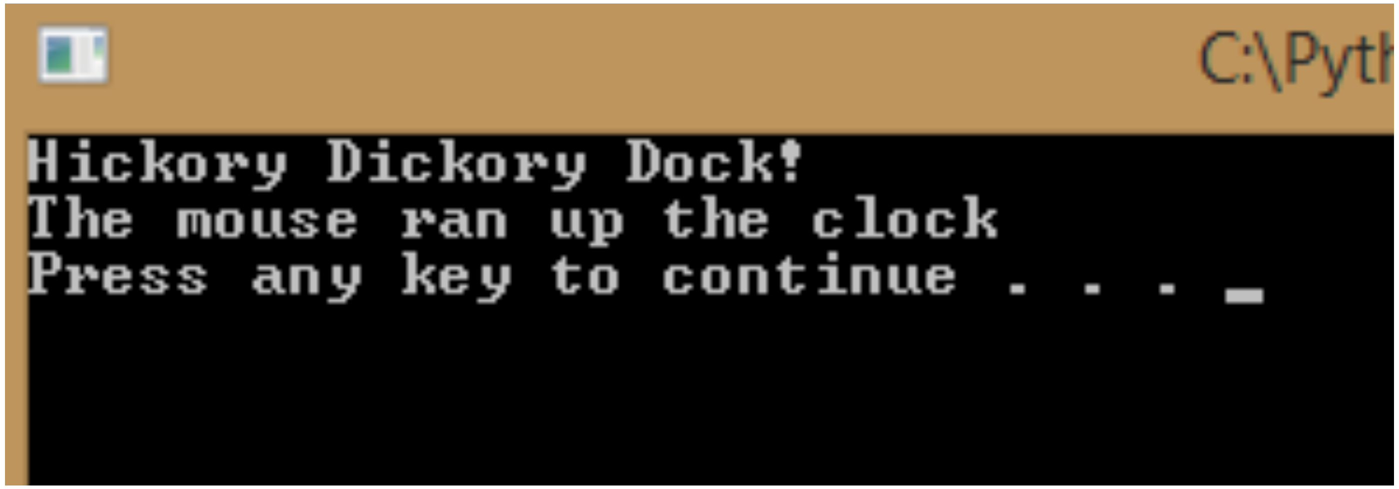
```
print('Hickory Dickory Dock!')
print('The mouse ran up the clock')
```



```
Hickory Dickory Dock!
The mouse ran up the clock
Press any key to continue . . . _
```

You can also use “\n” to force a new line

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



```
Hickory Dickory Dock!
The mouse ran up the clock
Press any key to continue . . . _
```

Here’s a neat Python trick: triple quotes!

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

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

When you put the string in triple quotes, it will be displayed the way you have the string in the text editor

DEMO

Line Management

Same problem, multiple solutions

Which do you think is better?

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

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

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

Geek Tips

- **There is often more than one way to solve the same problem**
- **Sometimes it really doesn't matter which way you do it, as long as it works!**

When good code goes bad...

There is another important programming concept you need to learn as well

- It’s okay to make mistakes in your code
- **All** programmers make typing mistakes and coding mistakes

So it might be useful to practice finding our mistakes

```
print(Hickory Dickory Dock)
print('It's a small world')
print("Hi there')
prnit("Hello World!")
```

```
print('Hickory Dickory Dock')
print("It's a small world")
print("Hi there")
print("Hello World!")
```

Your challenge should you choose to accept it

- Write a program that will display the following poem on the screen

There once was a movie star icon
who preferred to sleep with the light on.
They learned how to code
a device that sure glowed
and lit up the night using Python!

String variables and asking a user to enter a value

input

How can we ask a user for information?

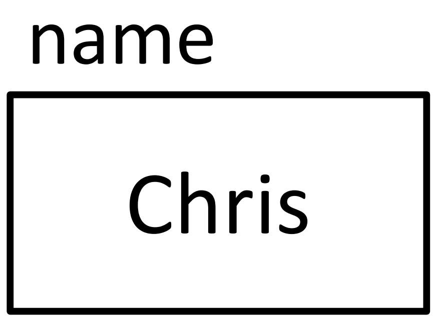
```
name = input("What is your name? ")
```

The **input** function allows you to specify a message to display and returns the value typed in by the user.

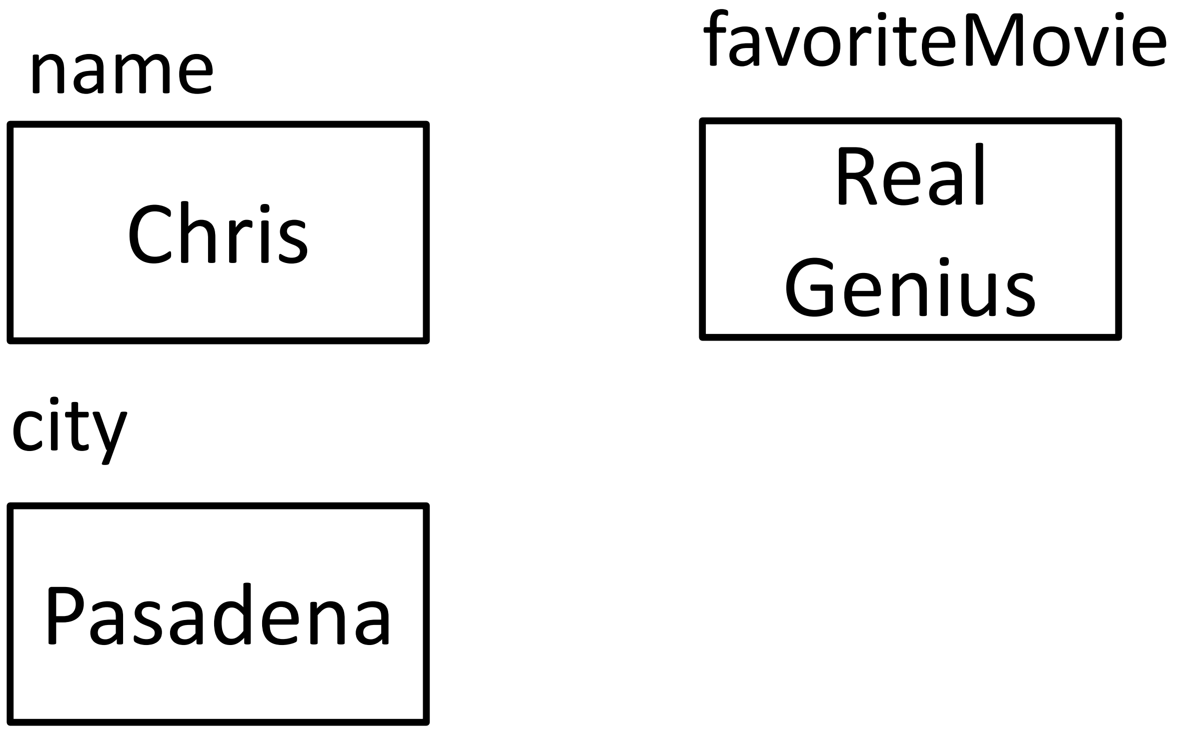
We use a variable to remember the value entered by the user.

We called our variable “name” but you can call it just about anything as long the variable name doesn’t contain spaces

Think of a variable as a box where you can store something and come back to get it later.

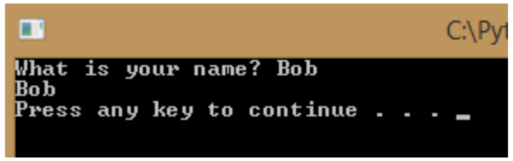


If you need to remember more than one value, just create more variables



You can access the value you stored later in your code

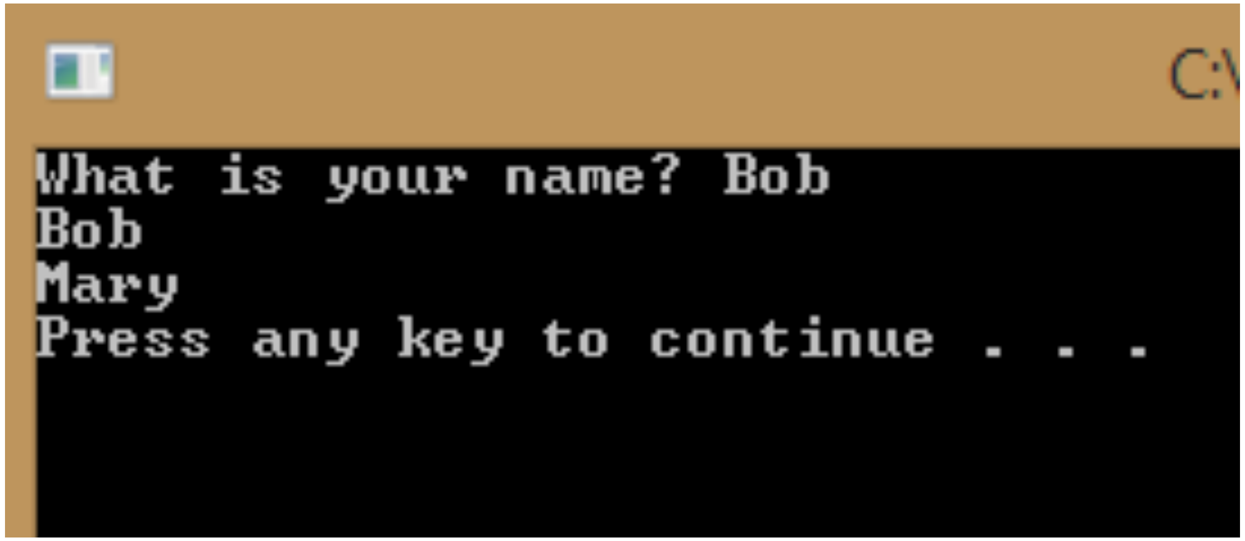
```
name = input("What is your name?
print(name)
                                   "
```



```
What is your name? Bob
Bob
Press any key to continue . . .
```

You can also change the value of a variable later in the code

```
name = input("What is your name?
print(name)
name = "Mary"
print(name)
```



```
What is your name? Bob
Bob
Mary
Press any key to continue . . .
```

Which of the following do you think would be good names for variables?

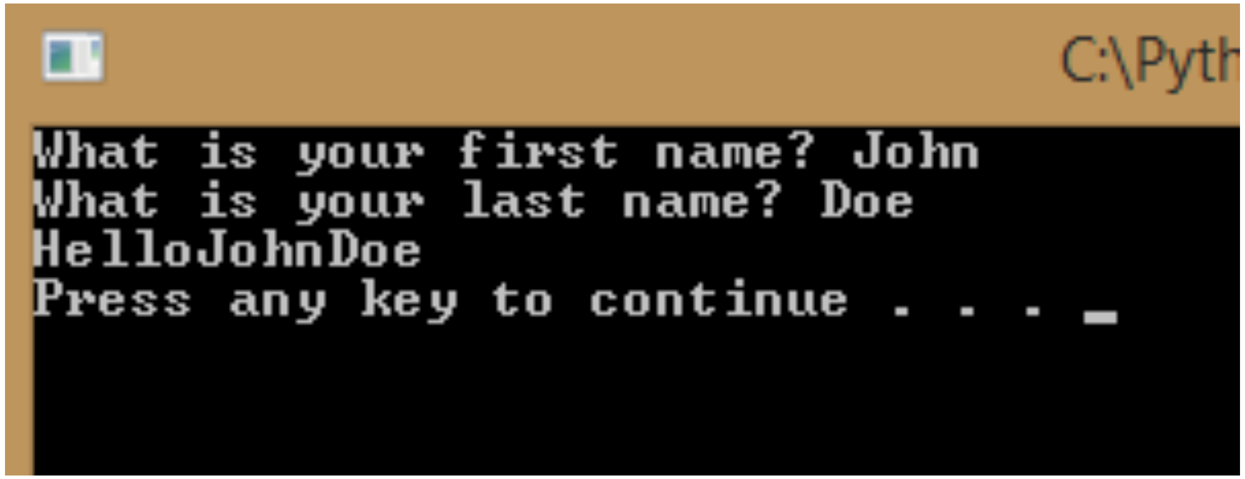
- Variable1
- First Name
- Date
- 3Name
- DOB
- DateOfBirth
- YourFavoriteSignInTheHoroscope

Variable names

- Should be meaningful (e.g. FirstName not variable1)
- Should be specific (BirthDate not Date)
- Should not 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 combine variables and strings with the + symbol

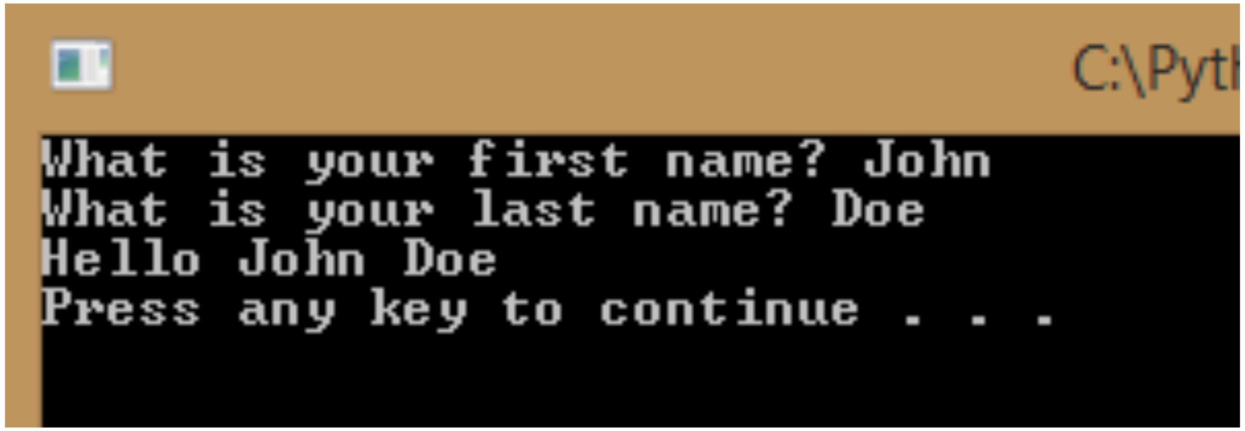
```python
firstName = input("What is your first name? ")
lastName = input("What is your last name? " )
print("Hello" + firstName + lastName)
```



```
What is your first name? John
What is your last name? Doe
HelloJohnDoe
Press any key to continue . . . _
```

Often you need to add punctuation or spaces to format the output correctly

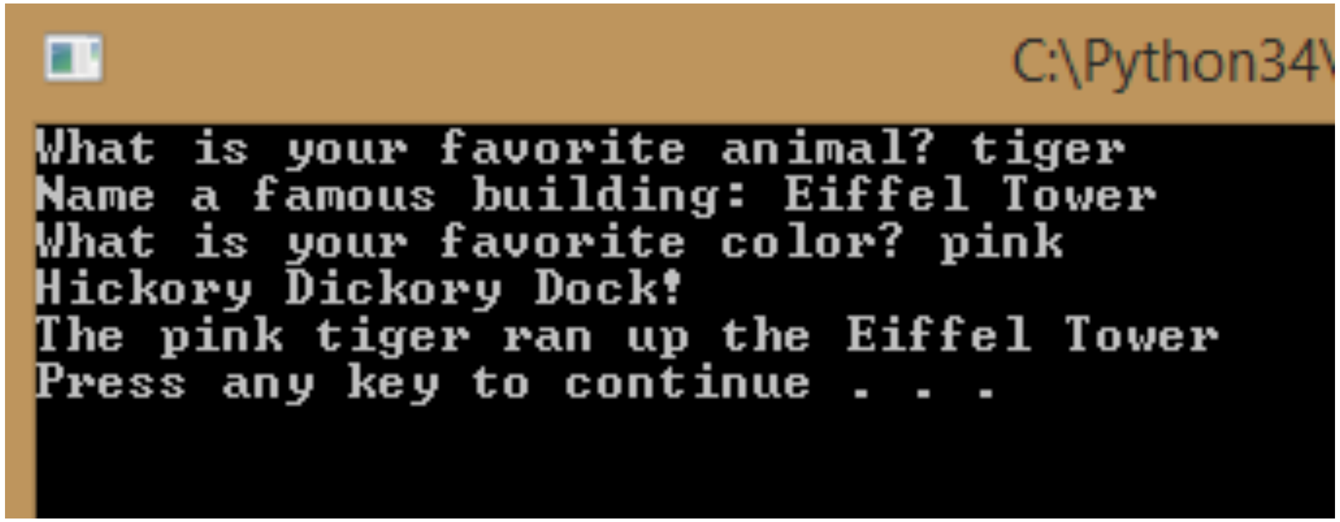
```python
firstName = input("What is your first name? ")
lastName = input("What is your last name? " )
print("Hello " + firstName + " " + lastName)
```



```
What is your first name? John
What is your last name? Doe
Hello John Doe
Press any key to continue . . .
```

Now you can create a story teller program!

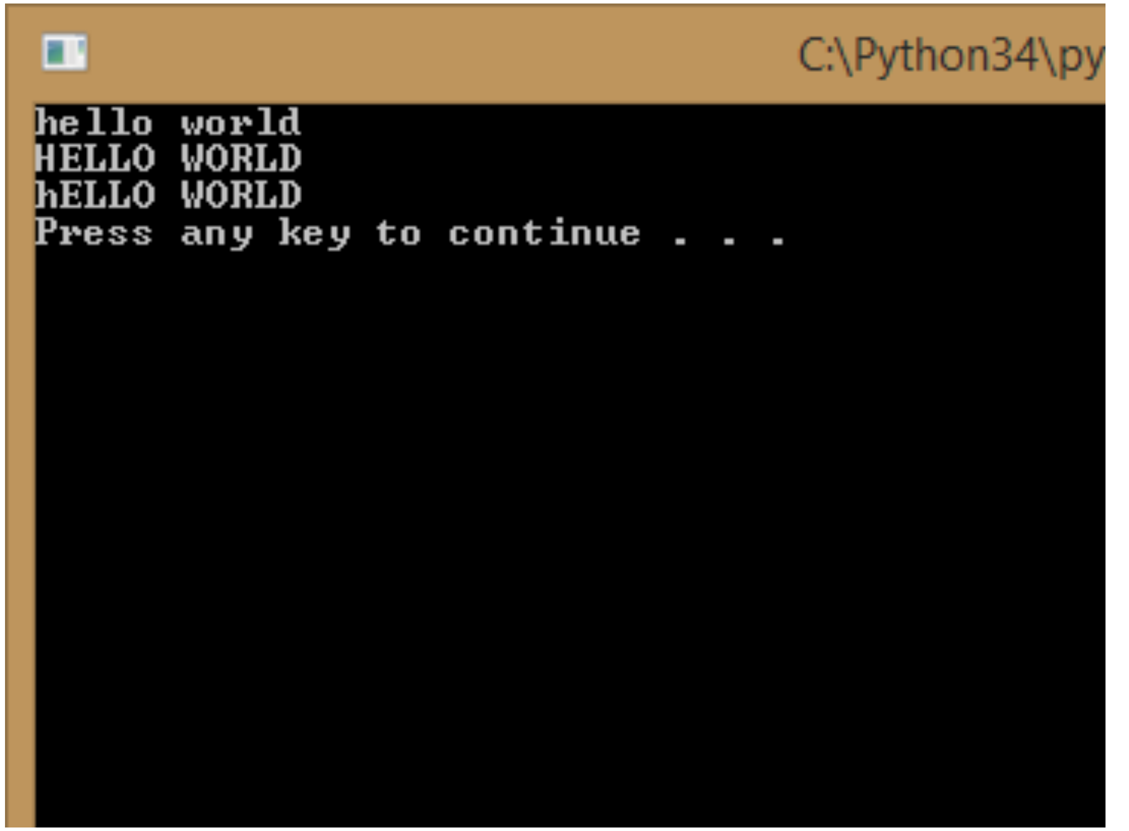
```python
animal = input("What is your favorite animal? " )
building = input("Name a famous building: ")
color = input("What is your favorite color? ")
print("Hickory Dickory Dock!")
print("The "+color+" "+animal+" ran up the "+building)
```



```
What is your favorite animal? tiger
Name a famous building: Eiffel Tower
What is your favorite color? pink
Hickory Dickory Dock!
The pink tiger ran up the Eiffel Tower
Press any key to continue . . .
```

Variables also allow you to manipulate the contents of the variable

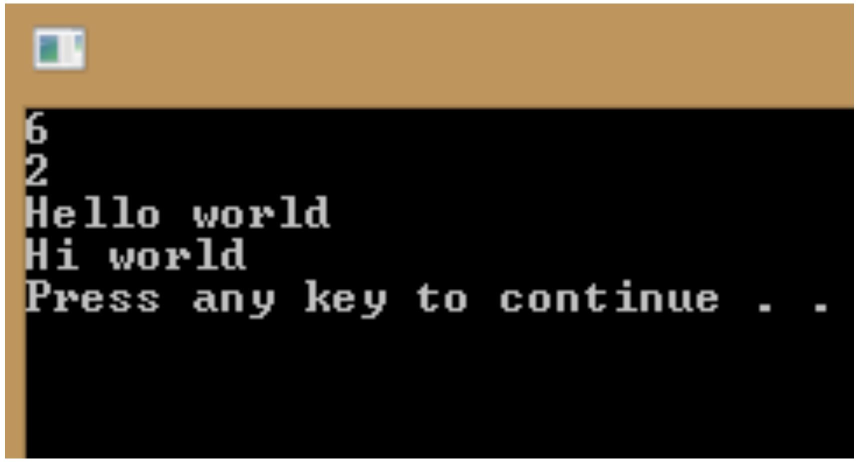
```
message = 'Hello world'
print(message.lower())
print(message.upper())
print(message.swapcase())
```



```
hello world
HELLO WORLD
hELLO WORLD
Press any key to continue . . .
```

What do you think these functions will do?

```python
message = 'Hello world'
print(message.find('world'))
print(message.count('o'))
print(message.capitalize())
print(message.replace('Hello','Hi'))
```



```
6
2
Hello world
Hi world
Press any key to continue . . .
```

How could we…

Have a user enter their postal code and then display that postal code in all upper case letters even if the user typed it in lowercase?

```
postalCode = input("Please enter your postal code: ")
print(postalCode.upper())
```

Did you notice?

The intellisense didn’t appear to help us select the **upper()** function.

That’s because our program didn’t know we were going to store a string value, in the postalCode variable. The **upper()** function is only for strings. A good habit when coding in any language is to initialize your variables. That means when you create them you give them an initial value.

```
postalCode = " "
postalCode = input("Please enter your postal code: ")
print(postalCode.upper())
```

How could we…

Ask someone for their name and then display the name someone with the first letter of their first and last name uppercase and the rest of their name lowercase?

```
name = " "
name = input("Please enter your name: ")
print(name.capitalize())
```

Functions and variables allow us to make new mistakes in our code…

Each line of code below has a mistake…

```
message = Hello world
23message = 'Hello world'
New message = 'Hi there'
print(message.upper)
print(mesage.lower())
print(message.count())
```

```
message = 'Hello world'
23message = 'Hello world'
New message = 'Hi there'
print(message.upper())
print(message.lower())
print(message.count('H'))
```

Storing numbers

You can store numbers in variables as well

```
age = 42
print(age)
```

With numbers you can do math

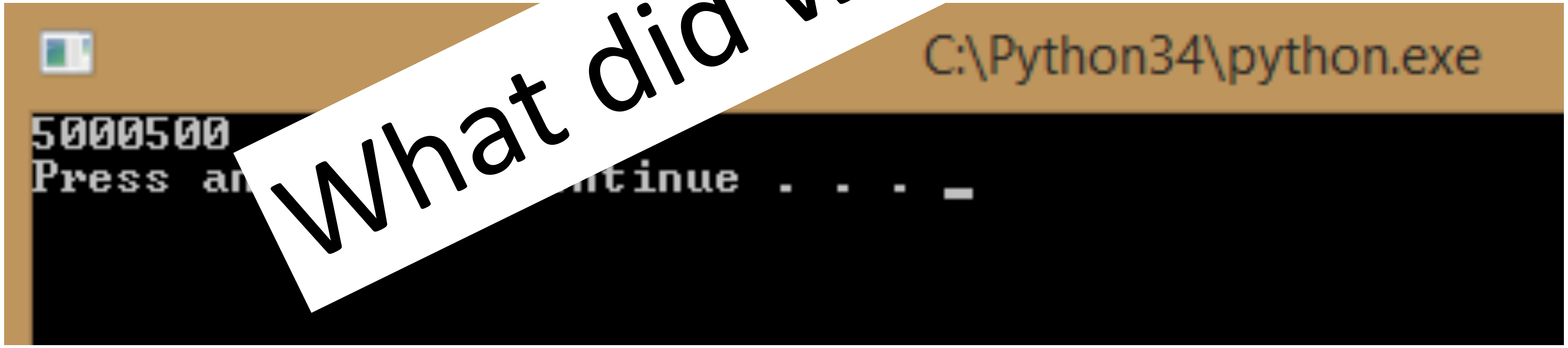
```
width = 20
height = 5

area = width * height
print(area)

perimeter = 2*width + 2*height
print(perimeter)

perimeter = 2*(width+height)
print(perimeter)
```

Math rules haven’t changed since Grade 4

```
Order of operations
( ) parentheses
**  exponent (e.g. **2 squared **3 cubed)
*/  multiplication and division
+ - addition and subtraction
```

Of course this means we have more ways to make mistakes too!

```python
salary = '5000'
bonus = '500'
payCheck = salary + bonus
print(payCheck)
```

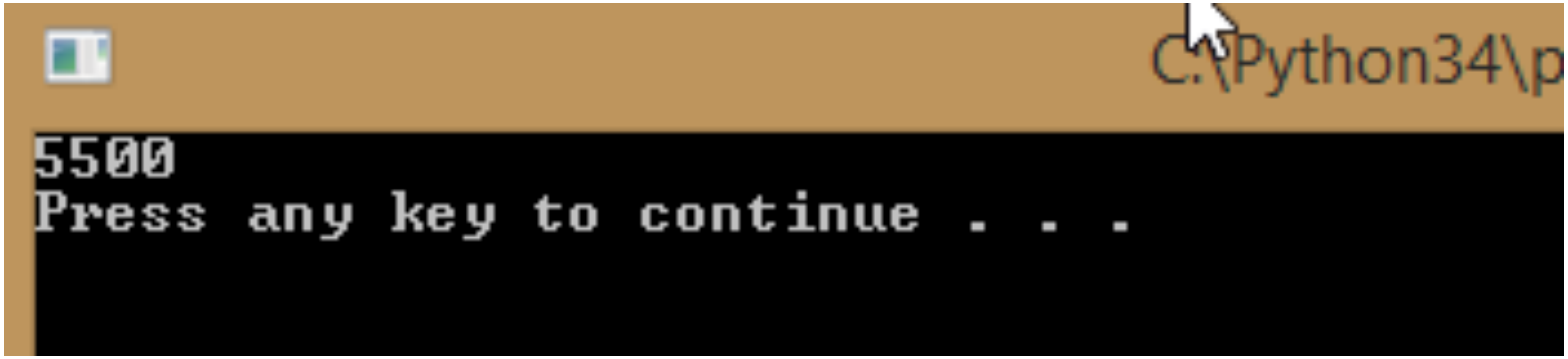


```
5000500
Press an           ntinue . . . _
```

What did we do wrong?

Because we put quotes around the values, the program thought salary and bonus were strings so it concatenated instead of adding

```
salary = 5000
bonus = 500
payCheck = salary + bonus
print(payCheck)
```

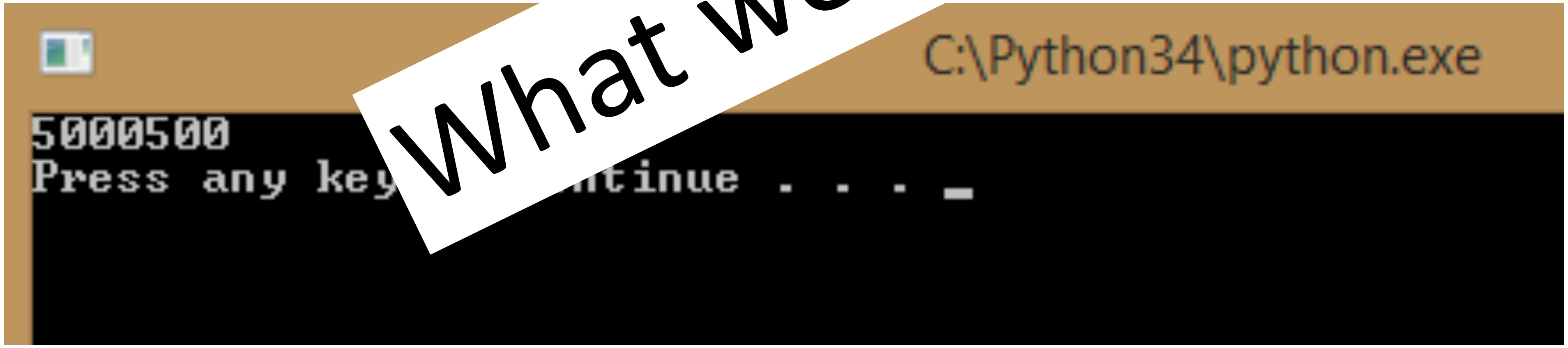


```
5500
Press any key to continue . . .
```

Could you ask the user to enter their bonus and salary values?

```
salary = input("Please enter your sal
bonus = input("Please enter your
payCheck = salary + bonus
print(payCheck)
```

What went wrong?



C:\Python34\python.exe

```
5000500
Press any key    ntinue . . . _
```

The input function always returns a string

We need a way to tell our program that it’s a number and not a string

There are functions to convert from one datatype to another.

```
int(value)    converts to an integer
long(value)   converts to a long integer
float(value)  converts to a floating number (i.e. a
number that can hold decimal places)
str(value)    converts to a string
```

Which function should we use for our scenario?

Since the amounts entered could include decimals – choose float

```
salary = input("Please enter your salary: ")
bonus = input("Please enter your bonus: ")
payCheck = float(salary) + float(bonus)
print(payCheck)
```

What do you think will happen if someone types “BOB” as their salary

The code crashes because we can’t convert the string “BOB” into a numeric value. We will learn how to handle errors later!

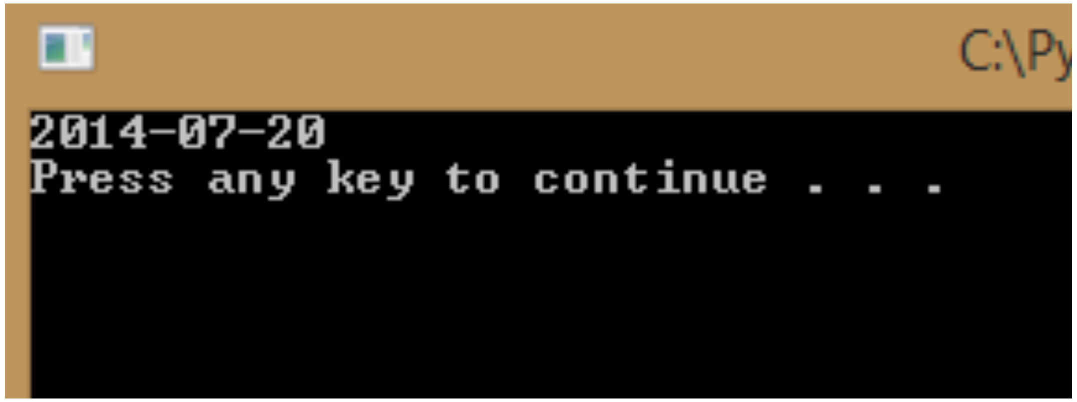
Working with dates and times

datetime

What is today’s date?

- The datetime class allows us to get the current date and time

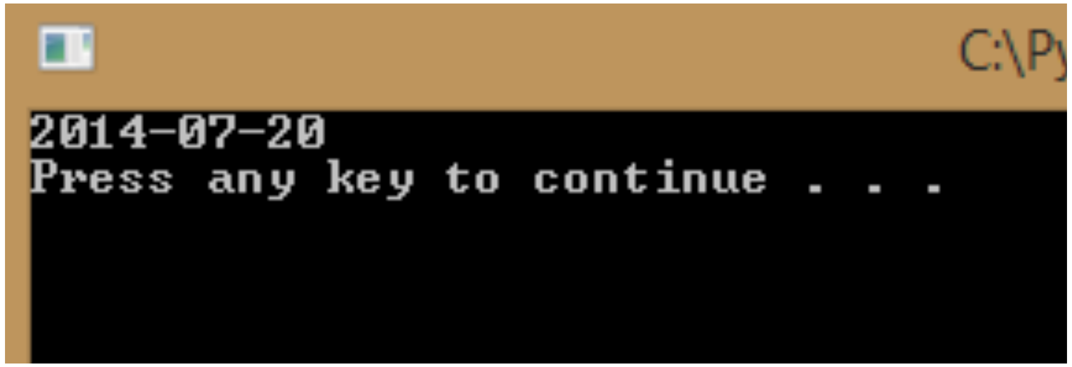
```
#The import statement gives us access to
#the functionality of the datetime class
import datetime
#today is a function that returns today's date
print (datetime.date.today())
```



```
2014-07-20
Press any key to continue . . .
```

You can store dates in variables

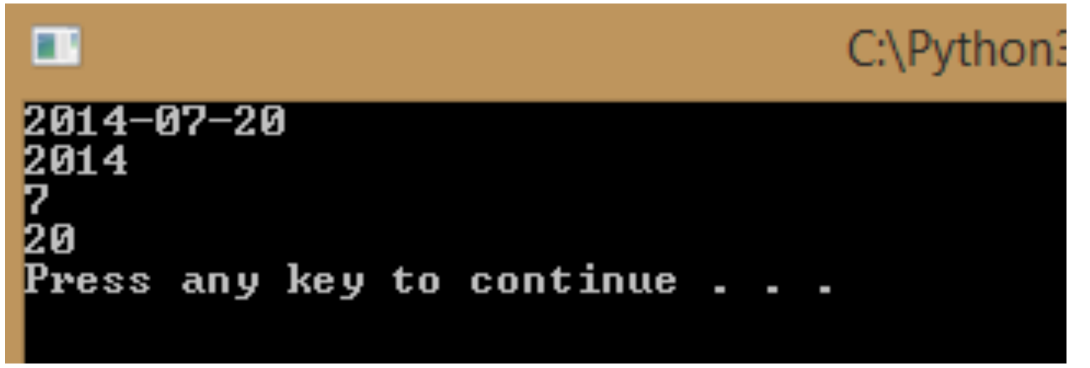
```python
import datetime

#store the value in a variable called currentDate
currentDate = datetime.date.today()
print (currentDate)
```



```
2014-07-20
Press any key to continue . . .
```

You can access different parts of the date

```python
import datetime
currentDate = datetime.date.today()
print (currentDate)
print (currentDate.year)
print (currentDate.month)
print (currentDate.day)
```



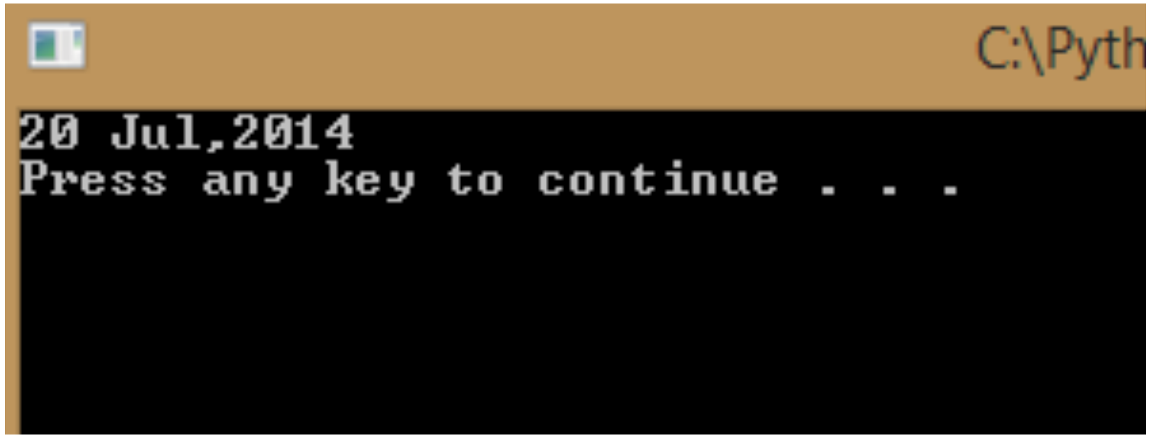
```
2014-07-20
2014
7
20
Press any key to continue . . .
```

But what if you want to display the date with a different format?

- Welcome to one of the things that drives programmers insane!
- Different countries and different users like different date formats, often the default isn’t what you need
- There is always a way to handle it, but it will take a little time and extra code
- The default format is YYYY-MM-DD

In Python we use strftime to format dates

```python
import datetime
currentDate = datetime.date.today()
#strftime allows you to specify the date format
print (currentDate.strftime('%d %b,%Y'))
```



```
20 Jul,2014
Press any key to continue . . .
```

What the heck are %d %b and %Y?

- %d is the day of the month
- %b is the abbreviation for the current month
- %Y is the 4 digit year

Here’s a few more you may find useful

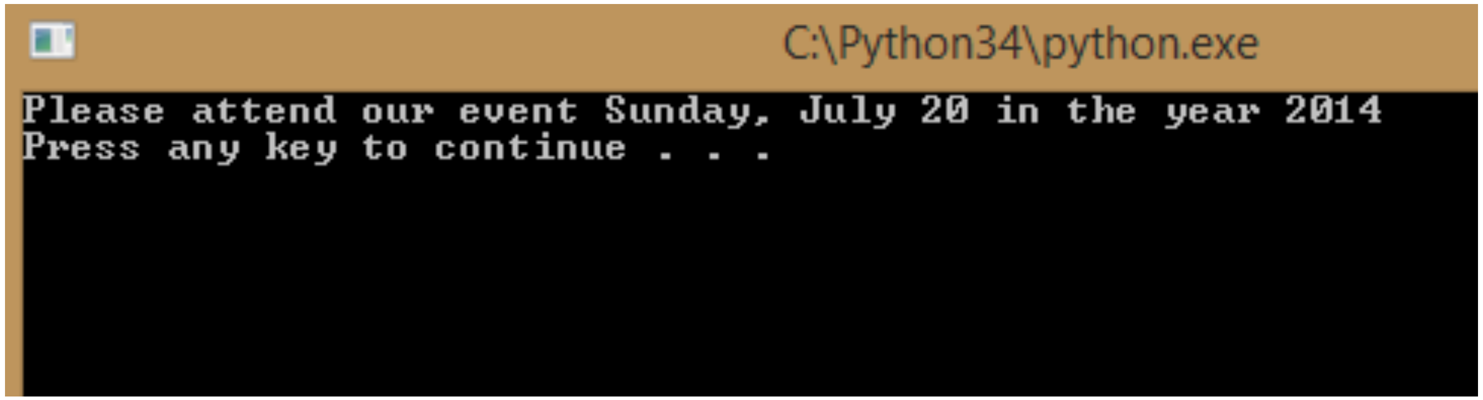
- %b is the month abbreviation
- %B is the full month name
- %y is two digit year
- %a is the day of the week abbreviated
- %A is the day of the week

- For a full list visit strftime.org

Could you print out a wedding invitation?

“Please attend our event Sunday, July 20th in the year 1997”

```
import datetime
currentDate = datetime.date.today()
#strftime allows you to specify the date format
print (currentDate.strftime
('Please attend our event %A, %B %d in the year %Y'))
```


C:\Python34\python.exe

```
Please attend our event Sunday, July 20 in the year 2014
Press any key to continue . . .
```

So… what if I don’t want English?

- In programmer speak we call that localization
- Did I mention dates drive programmers insane?
- By default the program uses the language of the machine where it is running
- But… since if you can’t always rely on computer settings it is possible to force Python to use a particular language
- It just takes more time and more code. You will probably want the babel Python library http://babel.pocoo.org/

Can I ask a user for their birthday?

```
birthday = input ("What is your birthday? ")
print ("Your birthday is " + birthday)
```

Can you think of any situations where this code might not work the way we want?

Can I ask a user for their birthday?

```
birthday = input ("What is your birthday? ")
print ("Your birthday is " + birthday)
```

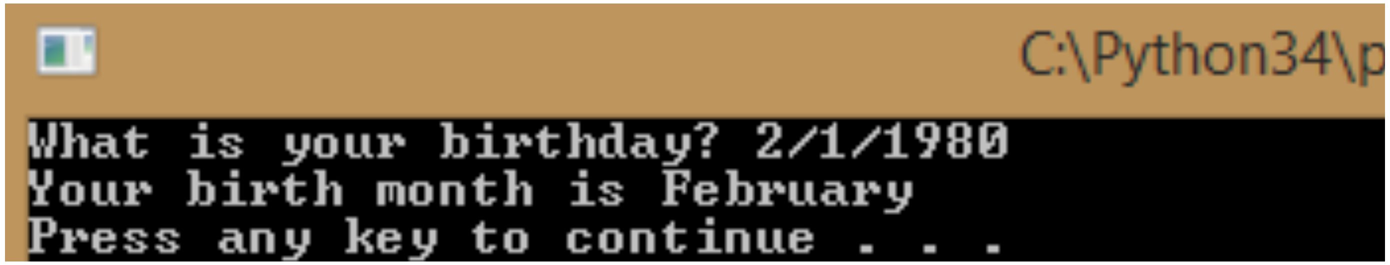
What datatype is birthday?

string

if we want to treat it like a date (for example use the datetime functions to print it in a particular format) we must convert it to a date

The strptime function allows you to convert a string to a date

```
import datetime
birthday = input ("What is your birthday? ")
birthdate =
datetime.datetime.strptime(birthday,"%m/%d/%Y").date()
#why did we list datetime twice?
#because we are calling the strptime function
#which is part of the datetime class
#which is in the datetime module
print ("Your birth month is " + birthdate.strftime('%B'))
```



```
What is your birthday? 2/1/1980
Your birth month is February
Press any key to continue . . .
```

But what if the user doesn’t enter the date in the format I specify in strptime?

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

- Your code will crash so…
- Tell the user the date format you want

```
birthday = input ("What is your birthday? (mm/dd/yyyy) ")
```

- Add error handling, which we will cover in a later module

Dates seem like a lot of hassle, is it worth it? Why not just store them as strings!

- You can create a countdown to say how many days until a big event or holiday

```
nextBirthday =
datetime.datetime.strptime('12/20/2014','%m/%d/%Y').date()
currentDate = datetime.date.today()
#If you subtract two dates you get back the number of days
#between those dates
print (nextBirthday - currentDate)
```

Dates seem like a lot of hassle, is it worth it? Why not just store them as strings!

- You can tell someone when the milk in their fridge will expire

```
currentDate = datetime.date.today()
#timedelta allows you to specify the time
#to add or subtract from a date
print (currentDate + datetime.timedelta(days=15))
print (currentDate + datetime.timedelta(hours=15))
```

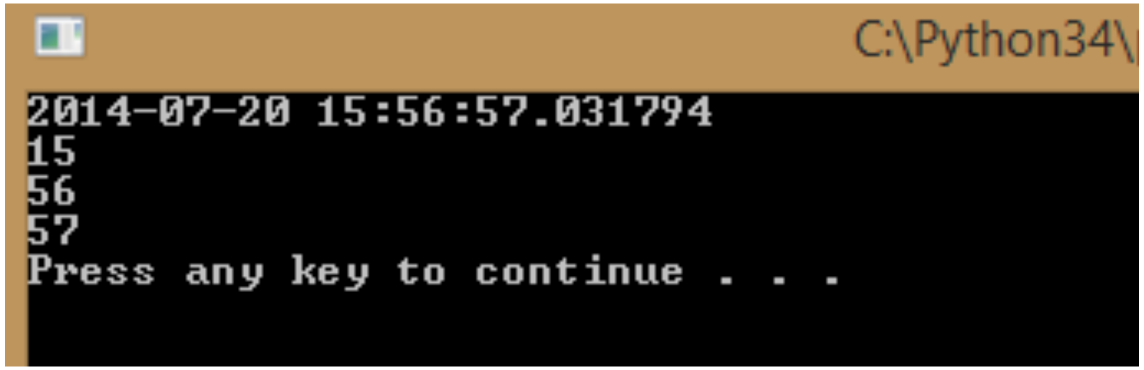
You will be amazed how often you need to work with dates!

- If datetime doesn’t have what you need, check out the dateutil library (for example you might want to know the number of years between two dates instead of number of days)

What about times?

- It is called Date**time**, so yes it can store times.

```
import datetime
currentTime = datetime.datetime.now()
print (currentTime)
print (currentTime.hour)
print (currentTime.minute)
print (currentTime.second)
```

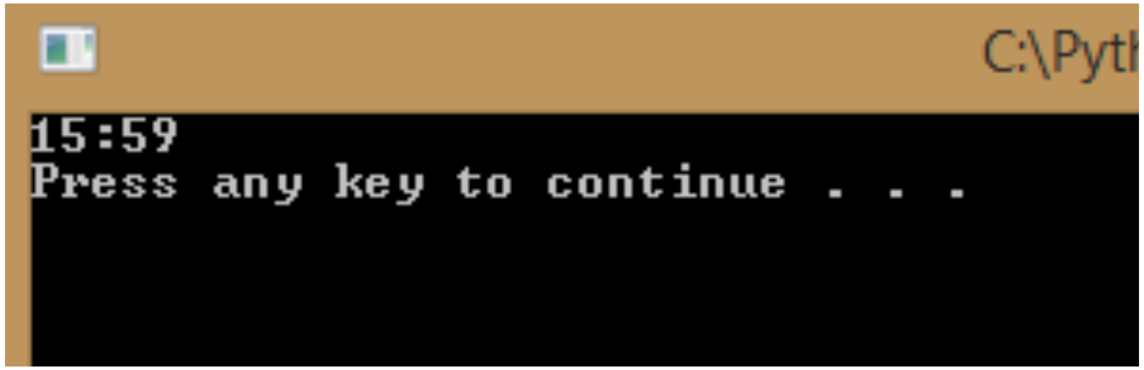


```
2014-07-20 15:56:57.031794
15
56
57
Press any key to continue . . .
```

Just like with dates you can use strftime() to format the way a time is displayed

```python
import datetime
currentTime = datetime.datetime.now()
print (datetime.datetime.strftime(currentTime,'%H:%M'))
```

%H	Hours (24 hr clock)
%I	Hours (12 hr clock)
%p	AM or PM
%m	Minutes
%S	Seconds



```
15:59
Press any key to continue . . .
```