Getting Started and Python Basics

Running python programs at the command line

- In a command windows (aka 'terminal' or 'shell' window)
- For a program in a file name myprogram.py type 'python myprogram.py'

Starting the python interpreter at the command line

- In a command windows (aka 'terminal' or 'shell' window) type 'python'
- The python command line prompt in IDLE is '>>>'
- At the command prompt you can enter things called *expressions* and *statements*.

Expressions and Statements

Expression:

Something which evaluates to a value. (e.g. 2+2)

Statement:

- A line of computer code that does something, that can be executed
- A computer program consists of one or more statements
- Statements can include expressions

The IDLE shell will automatically evaluate statements *and* expressions.

- IDLE will directly evaluate expressions and statements that are entered at the prompt.
- For example the statement a = 2+ 2 could be used as a line in a computer program.
  - Enter this at the prompt: a = 2+2
  - What happened?

- The following are examples of expressions - they are not complete statements. They could not be used by themselves as a line in a program. Try entering them at the prompt. What happens?
  - 2
  - 2 + 2
  - pow(2,3)

    - a
- remember that IDLE evaluates statements *and* expressions

Python has 33 keywords: the defining lexicon of the language.

- these words are built into the language and cannot be used for other purposes, in particular they cannot be used as variables
- 'False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield
- All but 'True','False', and 'None' use only lowercase.

Variable - a name that refers to a value

- all programming languages have some way to implement variables
- variables are assigned a value using an assignment operator
- in Python, and most programming languages, variable names must begin with a letter, but can also consist of letters and numbers after the first letter
- Python is case sensitive, so the variable name 'rover' is not the same as 'ROVER', and not the same as 'rOvEr'.

Assignment

- in python the equal sign, = , is the assignment operator
- the equal sign does not mean 'equal', rather it is a command to assign a value to a variable. (use two equal signs ==, to check equality, be careful not to confuse assignment with checking equality)
- enter: a = 1
- Now enter: a
- Now you can use a in places where you would like to use the value 1.
- enter: print(1)
- enter: print(a)

Basic Data Types in Python

- a piece of data is a value
- every value in python has a type (or a class)
- use the type() command to show the type of a value in python
- **int** (i.e. integer)
    - a number without any decimal point

    - enter: type (1)
- **float** (i.e. a floating point number)
    - floating point numbers are stored in a kind of scientific notation with a mantissa and an exponent
    - the exponent may or not be shown
    - floating point numbers cannot always be exact because of the way they are stored in computer memory
    - enter: 2.3
    - enter: 2.3e-7
    - enter: type (2.3)
- **bool** (i.e. a boolean value, can be either True or False)
    - enter 1==2 (with two equal signs, not 1)
    - enter: type(1==2)
    - enter: type(1=2) {i.e. just one equal sign}
        - why did you get an error?
- **string** ' a sequence of symbols (numbers, letters, punctuation) bracketed by quotation marks
    - you can use single or double quotes or triple quotes
    - double quotes are used for strings in these lectures
    - triple quotes are usually reserved for help text inside function definitions.
    - Enter: type ("hello")
    - Enter: print("hello")
    - Enter: print(hello)
        - Why did you get an error?
    - You can refer to positions in a string using slice notation, e.g. "hello"[1:3] returns the substring from position 1 up to, but not including 3.
        - IMPORTANT, python begins counting with 0.
- **list** - a list contains a series of elements, each of which can be any data type, enclosed in brackets and separated by commas
    - example [1,2.7,"hello"]
    - enter: type ([1,2.7,"hello"])
    - enter: type([1,2.7, "hello")
    - type ([1,2.7,"hello")
        - why did you get an error?
    - You can refer to positions in a string using slice notation, e.g. [1,2.7,"hello"][1:3] returns the sublist from position 1 up to, but not including 3.

        - IMPORTANT, python begins counting with 0.
- **Dictionary** – a special data type that can be used to look up information. A dictionary is enclosed by curly braces, {}. Each item in a dictionary includes a 'key' followed by a colon, :, followed by a value. The entries are separated by commas. A key must be an immutable (non-changeable) type, whereas values can be of any type.
    - example {1: "hello", 2: "goodbye"}
- python recognizes many values automatically and assigns them a type.
    - Numbers: e.g. 1
    - Strings: e.g. "hello"
    - Lists: e.g. ["a",1,3.7]
- Names that are not assigned a value do not automatically have a type.
    - Enter: a (i.e. without quotes)
    - Python does not know what a is

Operators

An operator is a pre-defined symbol that tells the interpreter to do a specific operation. They are the building blocks of most expressions and statements.

- Arithmetic operators, e.g. '+', '*', '-' and '/'
- The assignment operators: e.g. '=', '+='
- Comparison operators: e.g. '==', '>', '>='
- Membership and Identity operators: e.g. 'in', 'not in' , 'is' and 'is not'
- Many operators will work on many different data types, for example '+'
    - Enter: 2+2
    - Enter: "a" + "b"
    - Enter: "a" + " " + "b"
    - Enter : [1,-4.5, 'c'] + ["hello"]
    - Enter : [1,-4.5, 'c'] + "hello"
        - Why did this last example give you an error?

The help() function

- enter: help()
    - this starts running the program called 'help'

    - the ‘help’ program has its own command prompt, ‘help>’
    - hit return to leave the program and return to the command prompt
- try upper case, enter: Help()
    - All text is case sensitive in python
- ‘help’ is an example of a built-in function. All built-in python functions have lower-case names.

Writing a program

Go to ‘file’ and click ‘New Window’. This opens up another window that looks kind of like the IDLE interpreter window but instead is a text editor.

- Enter: ## this is my ‘hello world’ program
    - The '#' sign, or multiple '#' signs, tells the interpreter that the lines is just a comment and should be ignored
- Enter: print("hello world")
- Click on file, save, or hit cntrl + s, to save the file. Give it a name that explains a little bit of what it does.
- Always save python program files with the extension ‘.py’. If you don't they won't run. The '.py' extension tells the python interpreter that it is a python text file.
- On a windows machine, hit function key 5 (i.e. f5) to run the program.

Functions

- Functions are a type of python object. We already learned about some other types, like ‘integer’, ‘float’, ‘string’ and 'list'
- Python has many built in functions (i.e. instances of type 'function').
- Functions are often grouped into modules.
- New functions can be defined using the ‘def’ statement.
- To call a function (i.e. to get the function to do what it does), type the name of the function followed by parentheses. Depending on the function there may be arguments inside the parentheses.
- Functions ***ALWAYS*** have parentheses after them, even if empty.
- Example, the ‘pow’ function takes two arguments, each of which must be a number or a variable that has a numerical value.

- e.g. enter: pow(2,3)

Importing modules

- Modules are files that contain python functions. By importing a module into your program you get to use all of the functions that are in that module.
- The 'import' command is used to import a module.
- Example: enter: import os
- This will import the os module, which contains many functions that are used use features of the operating system that you are using.
- Enter: dir(os)
- Enter: help(os)

- For example, to find the current directory that you are in, use the 'getcwd()' function
- Enter: os.getcwd()

    - Notice the double backslashes in the string. These are there because backslashes ('\') are often used as part of special characters. For example a tab character is often coded as '\t' and an end-of-line character is coded '\n'. For this reason if you want to use an actual backslash as itself, and not as part of a special character, you need to precede it by another backslash.

- Try using the help command to get information on the getcwd function.

Write another program

- Go to file and click 'New Window'
- Enter: # a program to get the current directory and save the value in a string
- Enter: import os
- Enter: dirstring = os.getcwd()
- Save the file, and run it.
- What happened? Did any results print out? Why not?
- To see that IDLE now knows the value of 'dirstring', enter: dirstring

- Go back to the program and add one more line to the end:
  print(dirstring)

- Now run the program again.

The ‘help’ and ‘dir' functions

- The ‘dir()’ command lists all of the methods and attributes that are associated with an object.
- enter: dir(os)
- try some other examples, e.g. 'dir(1)' 'dir(dir)' dir('hello')
- What do the results tell you about the dir function?

- What you see is a listing of all the things associated with the os module.

- If you just type ‘dir()’ you will get a listing of everything that you have loaded into the interpreter.

- The ‘help()’ command returns documentation that is associated with objects.

    - enter: help(1)
    - enter: help(dir)

The ‘type()’ command
We already used this, but it is good to remember. Sometimes we lose track of just what type something is. If you are unsure, or a variable is not behaving the way it should, use the type() command to check its type.

Working with strings

- The simplest string is an empty string, "" (two adjacent double quotes)
- To see the attributes of a string
- Enter: dir("")
- What does this tell you?

- For example to change the case of a string, you can use the swapcase function that is an attribute of all strings.
- Enter: a = 'teststring'
- Enter: a.swapcase()

- You don't even have to use a variable. In python you can chain expressions to make a larger expression.
- Enter: "teststring".swapcase()

- A string is kind of like a list of characters.
- To pull out a certain character of a string, as a string use brackets.
- Enter: a[2]

- You can pull out a subset of characters using the slice notation
- Enter: a[2:6]

Working with lists

- Lists are data types that are bracketed by parentheses
- They can contain pretty much any other data type, including lists.
- You can access any element of a list using a bracket notation (like with a string), and you can access a slice of a list.
- mylist = ["hello", "world", a]

- Try combining you're list named mylist and the list [1,2,3] using the '+' operator.

- Python creates a single new list every time you execute the [] expression, and Python never creates a new list if you assign a list to a variable.
    - A = B = [] # both names will point to the same empty list
    - A = []; B = [] # independent lists
- Try the following
    - create a short list and assign it to the variables a and b:
        - A= B = ["hello", "world"]
    - assign the zero element of a to "goodbye"
        - A[0] = "goodbye"
    - Now check the value of A and B.

    - Notice that because B is assigned to A, it returns the same value.
- Now make two independent lists
    - Create a short list and assign it to A
        - A = ["hello", "world"]
    - Create an identical list and assign it to B
        - B = ["hello", "world"]
    - assign the zero element of A to "goodbye"
        - A[0] = "goodbye"
    - Now check the value of A and B.
    - Notice that because B is not assigned to A, it returns the original value of B and not what A is now.

Working with Dictionaries

- Make an empty dictionary
    - a = {}
- add an item by calling the dictionary with [] enclosing a new key and assigning a value to be associated with that key
    - a[1] = “car”
    - now the value of a is {1: 'car'}
- Make a dictionary with some items in it
    - b={"friend": "John", "girlfriend":"Mabel","family":["mom","dad","sister","brother"]}
    - notice that a key can be any immutable type, such as a number or a string, and a value can be any type. Try adding a new item.
    - b[1] = False
    - now check the value of b
- There is no order to the items in a dictionary and you cannot reference the items as if they are items in a list or a string.

Python is an ‘object-oriented’ language

- we can think of everything in python as an object
- variables are objects
- every object has 3 things:
    - a type or class
        - enter: type(a)
    - content (i.e. the value of the variable)
        - enter: a
    - a unique identity (the place where the variable is in computer memory)

        - we can find the identity of an object by using the 'id' command
        - enter: id(a)
    - the names of variables are maintained separately in a part of memory set up by the program and called the 'namespace'
    - The identity and the type of objects *cannot* be changed
        - If you think you have changed the identity and type what has actually happened is that the old object was destroyed and a new one, with the same name has been created.
    - Depending on the type of an object, it may be possible to change the content (i.e. the value).

Assignment for today :Write a program
Close IDLE and open it again. This will cause any modules that were loaded and any variables you used to be lost. Alternatively, click on 'Shell' and 'restart shell'.
Open a new file and save it in your week1 directory.
After the program is done and runs, and does everything it should, turn it in under the Assignment for Week1 on the Canvas site.

This program will do lots of simple python things. Be sure to include comments in your program file.

What your program should do:

1. Addition and printing:
    - add up the numbers from 1 to 10 and print the result to the screen
    - add up 10 strings, each with a single, different letter in it, to make one string, and print the result to the screen
    - make two lists and try adding them together and print the result.
2. Using dir, assignment, type() and print()
    - call the dir() function for a list (e.g. use: dir([])) and assign a variable to what the dir() function returns.
    - Print the type of this variable to the screen using the print function and the type() function (it should be a list)
    - Print this variable to the screen
3. Working with lists
    - 3.1

        - Using the list from part 2, print the length of the list to the screen (use the len()) function
        - Do a sort of the list using the sort() function that comes with lists.
        - Print the sorted list to the screen followed on the same line by the number of items in the list
        - Do a reverse sort of the list using the sort() function that comes with lists.
        - Print the sorted list to the screen followed on the same line by the number of items in the list
        - Append to the list the string: "test" and print the list to the screen
    - 3.2
        - Make a variable of a new list of the numbers from 1 to 100 using list comprehension (look it up) and print the list to the screen
            - Make a new list that has the 39th element up to but not including the 70th element by taking a slice of the list and print the list
            - In this new list, find the position of the value 45 using the index() method that belongs to lists. Print this position and the value in the list at this position.
            - Make a new list that has the last 10 values of the 100 element list by using slice notation with negative values and print the list
    - 3.3
        - Create a variable that is a string with your name in it,
            - e.g. a = "Jody Hey"
        - Make a new list of the symbols in this string by using the list() function
        - Print the list
        - Sort the list, and print the sorted list.
4. Working with strings
    - 4.1
        - Import the math module and make a variable that is a string of math.pi using the str() function
        - Print the type of math.pi and the type of the variable you created using str()
    - 4.2

- Make a variable that is a string with your name with a space separating the first and last names
- Print this variable and an upper case version of the string using the upper() method.
- Print a slice of this string that includes the 2nd thru 5th characters
- Make a list of the two parts of your name, by using the split() method that belongs to strings and print the list.

5. Working with dictionaries
    - Create a dictionary using curly braces that has as values the names of 5 of your friends and as keys your nicknames for them (you can't make up names if you like).
    - Print the keys of the dictionary
    - Print the values of the dictionary
    - Print the items in the dictionary
    - Make another dictionary with the days of the week as values and integers 0 thru 6 as keys.
    - Print the items in the second dictionary.
    - Create a new dictionary by adding the two previous dictionaries. Print this dictionary.
    - To this last dictionary add another dictionary that you've made using the key "XX" and the value "junk" and print the new dictionary.
6. Using functions in a module
    - Get the week1module.py program from Canvas and put it in the same directory where your program is.
    - In your program import this module
    - In your program, create a list of positive numbers (any numbers, as many as you want)
    - Edit the logsum() function to return, rather than print, the log of the sum.
    - Create a variable that has the log of the sum of the values by calling the logsum() function in week1module by passing to it this list of numbers.
    - Print the list and the log of the sum
    - In your program create a list of strings (any strings, as many as you want).

    - Create a variable that is a string with the words of the list joined by spaces by calling the joinwords() function in the week1module by passing to it this list of strings.
    - Print the list and the string of joined strings.
7. Using the os module
    - Import the os module and use it to get the name of the current directory. Assign a variable to this directory name.
    - Use the os.chdir() function to make the current directory be the root directory of your current hard drive.
    - Figure out how to get a listing of all the files that are contained in the current directory. (use help() and/or dir() to find any function that belong to the os module that can be used for listing the contents of a directory), and print the listing to the screen.
    - Use the os.chdir() function to return to the original directory (using the saved name).
    - Print the value returned by os.chdir()
8. Getting input from users
    - Write a line of code that instructs the user to type a number as input from the python command line (read about the input()) function)
    - Create a floating point variable from that number.
    - print the type of the variable (use type()) and the value of it to the screen
    - Write a line of code that instructs the user to type a sentence and assigns a variable to that sentence.
    - Print the sentence.

Be sure to run your program to make sure it does everything that it is supposed to do. The instructor should be able to just run it and have all the correct output printed to the screen.

Some things that may help:

- Put print statements in your code for each part of the assignment (e.g. before you do part 3.2 print("part 3.2")
- Try things out at the python command line before putting them in your program.

- use the dir() and help() functions to find out about what belongs to things and about how they work. Also don't hesitate to look things up on the internet.
- Be sure to include comments in your code so that a person (i.e. me) reading the file can understand what you are doing

