Python Basics

The Python Programming language

- a "high level" language
  - far removed from machine code
- source code is highly readable by people
- designed for short development time (i.e. fast program development)
  - core syntax (the set of basic language elements) is small
  - the standard library (set of precompiled functions) is large and comprehensive
- general purpose (suitable for all sorts of different problems)

Python History:

- Started in 1991 (and still overseen) by Guido van Rossum
- Has grown steadily in use
- Still being developed and changed
  - Often times you will read something about using python, and it will turn out that it is for an older or a newer version of the language than you are using.
  - The latest version of Python is 3.6
  - We will use version 2.7, as not all of the popular modules (libraries) are current with the latest version of Python.
- Used in many different fields, including bioinformatics

The process of 'installing' a programming language means that you install a program that can convert source code into commands the computer can understand.

Source code is a text file (usually written by a person) in a specific computer language (e.g. Python, C, Java)

Traditionally there are two types of source-code to machine-code converters:

- compiler. A compiler reads source code and produces an executable file that can be run on its own
  - once a program is compiled, it can be run without having the compiler program

- interpreter. An interpreter reads the source code and then generates machine code and then runs the machine code.
    - When using an interpreter, 'running' a program means using the interpreter
    - Python is an interpreted language
    - To run a python program you must have the python interpreter on your machine
- There are lots of exceptions and ways around this distinction

The standard Python installation includes an integrated development environment (IDE), which includes:

- a user interface (a shell that is called IDLE) that serves as text editor and debugger (debugger may not work on macs)
- the Python interpreter

There are many IDE's for python that are free or available for purchase and that are more powerful than IDLE. Some that I have used include:

Directory structure and file naming:

On your laptop, create a folder for this week's work.

For all python program files be sure to put '.py' on the end!.

Some rules to follow for your computer:

1. use lowercase letters when naming folders and files. Operating systems differ in how much they are case sensitive. Most bioinformaticians end up using multiple operating systems, so to avoid confusion they fall back on just naming things with lower case letters.
2. When naming folders and files, is us usually best to avoid introducing spaces
3. For the folder where your work will be, (e.g. ../bioinformatics) you want your file viewer to show you all parts of the names of the file and not to hide file extensions. In windows this means going into windows explorer, clicking on 'organize' and then 'folder and search options' and

then on ‘view’ and then clicking on then ‘Show hidden files, folders and drives’

4. Back up your workshop folder after every time you work in it. For example to a jump drive.

Using IDLE as a command line interface for python statements

The command prompt in IDLE is ‘>>>’
At the command prompt you can enter things called *expressions* and *statements*.
All text is case sensitive in python:

- 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()
- ‘help’ is an example of a built-in function. All built-in python functions have lower-case names.

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?
- **list** - a list contains a series of elements, each of which can be any data type, enclosed in brackets and separated by commas
  - example [1,2.3,"hello"]
  - enter: type ([1,2.3,"hello"])
  - enter: type([1,2.3, "hello")
  - type ([1,2.3,"hello")
    - why did you get an error?

- **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

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

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

Python has 31 keywords – these words are built into the language and cannot be used for other purposes, in particular they cannot be used as variables

and del from not while
as elif global or with
assert else if pass yield

break except import print
class exec in raise
continue finally is return
def for lambda try

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?

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
- Statements can include expressions
- We have already used several statements:
    - Assignment statement (i.e. a = 1)
    - Print statement
    - type() statement

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

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.