# Programming For Biology 2024

programmingforbiology.org

# Python 1

## Python Overview

Python is a scripting language. It is useful for writing medium-sized scientific coding projects. When you run a Python script, the Python program will generate byte code and interpret the byte code. This happens automatically and you don't have to worry about it. Compiled languages like C and C++ will run much faster, but are much much more complicated to program. Languages like Java (which also gets compiled into byte code) are well suited to very large collaborative programming projects, but don't run as fast as C and are more complex that Python.

Python has

- data types
- functions
- objects
- classes
- methods

**Data types** are just different types of data which are discussed in more detail later. Examples of data types are integer numbers and strings of letters and numbers (text). These can be stored in variables.

**Functions** do something with data, such as a calculation. Some functions are already built into Python. You can create your own functions as well.

**Objects** are a way of grouping a set of data and functions (methods) that act on that data.

**Classes** are a way to encapsulate (organize) variables and functions. Objects get their variables and methods from the class they belong to.

**Methods** are just functions that belong to a class. Objects that belong to the a class can use methods from that class.

## Running Python

There are two versions of Python: Python 2 and Python 3. We will be using 3. This version fixes some of the problems with Python 2 and breaks some other things. A lot of code has already been written for Python 2 (it's older), but going forwards, more and more new code development will use Python 3.

### Interactive Interpreter

Python can be run one line at a time in an interactive interpreter. You can think of this as a Python shell. To launch the interpreter, type the following into your terminal window:

```
$ python3
```

Note: '$' indicates the command line prompt. Recall from Unix 1 that every computer can have a different prompt!

First Python Commands:

```
>>> print("Hello, PFB!")
Hello, PFB!
```

> Note: `print` is a function. Function names are followed by (), so formally, the function is `print()`

## Python Scripts are Text Files

- The same code from above is typed into a text file using a text editor.
- Python scripts are always saved in files whose names have the extension '.py' (i.e. the filename ends with '.py').
- We could call the file `hello.py`

File Contents:

```
print("Hello, PFB!")
```

## Running Python Scripts

Typing the Python command followed by the name of a script makes Python execute the script. Recall that we just saw you can run an interactive interpreter by just typing `python3` on the command line.

Execute the Python script like this (% represents the prompt)

```
% python3 hello.py
```

This produces the following result in the Terminal:

```
Hello, PFB!
```

## A quicker/better way to run python scripts

If you make your script executable, you can run it without typing `python3` first. Use `chmod` to change the permissions on the script like this

`chmod +x hello.py`

You can look at the permissions with

```
% ls -l hello.py
-rwxr-xr-x  1 sprochnik  staff  60 Oct 16 14:29 hello.py
```

The first 10 characters you see displayed on the line have special meanings. The first character (`-`) tells you what kind of file `hello.py` is. `-` means a normal file, `d` a directory, `l` a link. The next nine characters come in three sets of three. The first set refers to the your permissions, the second set your group's permissions, and the last set to everyone else. Each three character set shows in order `rwx` for read, write, execute. If someone doesn't have a permission, a `-` is displayed instead of a letter. The three 'x' characters means anyone can execute or run this script.

We also need to add a line at the beginning of the script that tells the shell to run python3 to interpret the script. This line starts with `#`, so it looks like a comment to python. The `!` (exclamation mark or bang) is important as is the space between `env` and `python3`. The program `/usr/bin/env` looks for where `python3` is installed and runs the script with `python3`. The details may seem a bit complex, but you can just copy and paste this 'magic' line.

The file hello.py now looks like this

```
!/usr/bin/env python3
print("Hello, PFB!")
```

Now you can simply type the symbol for the current directory `.` followed by a `/` and the name of the script to run it. Like this

```
% ./hello.py
Hello, PFB!
```

# Syntax

## Python Variable Names

A Python variable name is a name used to identify a variable, function, class, module, or other object. A variable name starts with a letter, `A` to `Z` or `a` to `z`, or an underscore (`_`), followed by zero or more letters, underscores, and digits (`0` to `9`).

Python does not allow punctuation characters such as `@`, `$`, and `%` within a variable name. Python is a case sensitive programming language. Thus, `seq_id` and `seq_ID` are two different variable names in Python.

## Naming conventions for Python Variable Names

- The first character is lowercase, unless it is a name of a class. Classes should begin with an uppercase characters (ex. `Seq`).

- Private variable names begin with an underscore (ex. `_private`).
- Strong private variable names begin with two underscores (ex. `__private`).
- Python language-defined special names begin and end with two underscores (ex. `__file__`).

Picking good variable names for the objects you name yourself is very important. Don't call your variables things like `items` or `my_list` or `data` or `var`. Except for where you have a very simple piece of code, or you are plotting a graph, don't call your objects `x` or `y` either. All these name examples are not decriptive of what kind of data you will find in the variable or object. Worse is to call a variable that contains gene names as `sequences`. Why is this such a bad idea? Think about what would happen if you filled your car up at a store labelled 'gas station' that sold lemonade. In computer science, names should always accurately describe the object they are attached to. This reduces possibility of bugs in your code, makes it much easier to understand if you come back to it after six months or share your code with someone, and makes it faster to write code that works right. Even though it takes a bit of time and effort to think up a good name for an object, it will prevent so many problems in the future!

## Reserved Words

The following is a list of Python keywords. These are special words that already have a purpose in python and therefore cannot be used as variable names.

```
and         exec        not
as          finally     or
assert      for         pass
break       from        print
class       global      raise
continue    if          return
def         import      try
del         in          while
elif        is          with
else        lambda      yield
except      list        hash
```

## Lines and Indentation

Python denotes a block of code by lines with the same level of indentation. This keeps lines of code that run together organized. Incorrect line spacing and/or indention will cause an error or can make your code run in a way you don't expect. You can get help with indentation from good text editors or Interactive Development Environments (IDEs).

The number of spaces in the indentation need to be consistent, but a specific number is not required. All lines of code, or statements, within a single block must be indented the same amount. For example, using four spaces:

```
!/usr/bin/env python3
message = '' # make an empty variable
for x in (1,2,3,4,5):
    if x > 4:
        print("Hello")
        message = 'x is big'
    else:
        print(x)
        message = 'x is small'
    print(message)
print('All Done!')
```

## Comments

Including comments in your code is an essential programming practice. Making a note of what a line or block of code is doing will help the writer and readers of the code. This includes you!

Comments start with a pound or hash symbol `#`. All characters after this symbol, up to the end of the line are part of the comment and are ignored by Python.

The first line of a script starting with `#!` is a special example of a comment that also has the special function in Unix of telling the Unix shell how to run the script.

```
!/usr/bin/env python3

 this is my first script
print("Hello, PFB!") # this line prints output to the screen
```

## Blank Lines

Blank lines are also important for increasing the readability of the code. You should separate pieces of code that go together with a blank line to make 'paragraphs' of code. Blank lines are ignored by the Python interpretor.

# Data Types and Variables

This is our first look at variables and data types. Each data type will be discussed in more detail in subsequent sections.

The first concept to consider is that Python data types are either immutable (unchangeable) or not. Literal numbers, strings, and tuples cannot be changed. Lists, dictionaries, and sets can be changed. So can individual (scalar) variables. You can store data in memory by putting it in different kinds of variables. You use the `=` sign to assign a value to a variable.

## Numbers and Strings

Numbers and strings are two common data types. Literal numbers and strings like this `5` or `'my name is'` are immutable. However, their values can be stored in variables, which can be changed.

For Example:

```
gene_count = 5
 change the value of gene_count
gene_count = 10
```

> Recall the section above on variable and object names (and variables are objects in Python).

Different types of data can be assigned to variables, i.e., integers (`1`, `2`, `3`), floats (floating point numbers, `3.1415`), and strings (`"text"`).

For Example:

```
count   = 10     # this is an integer
average = 2.531    # this is a float
message = "Welcome to Python" # this is a string
```

`10`, `2.531`, and `"Welcome to Python"` are singular (scalar) pieces of data, and each is stored in its own variable.

Collections of data can also be stored in special data types, i.e., tuples, lists, sets, and dictionaries. You should always try to store like with like, so each element in the collection should be the same kind of data, like an expression value from RNA-seq or a count of how many exons are in a gene or a read sequence. Why do you think this might be?

## Lists

- Lists are used to store an ordered, *indexed* collection of data.
- Lists are mutable: the number of elements in the list and what's stored in each element can change
- Lists are enclosed in square brackets and items are separated by commas

```
[ 'atg' , 'aaa' , 'agg' ]
```

| Index | Value |
|---|---|
| 0 | atg |
| 1 | aaa |
| 2 | agg |

> The list index starts at 0

## Tuples

- Tuples are similar to lists and contain ordered, *indexed* collections of data.
- **Tuples are immutable: you can't change the values or the number of values**
- A tuple is enclosed in parentheses and items are separated by commas.

```
( 'Jan' , 'Feb' , 'Mar' , 'Apr' , 'May' , 'Jun' , 'Jul' , 'Aug' , 'Sep' , 'Oct' , 'Nov' , 'Dec' )
```

| Index | Value |
|---|---|
| 0 | Jan |
| 1 | Feb |
| 2 | Mar |
| 3 | Apr |
| 4 | May |
| 5 | Jun |
| 6 | Jul |
| 7 | Aug |
| 8 | Sep |
| 9 | Oct |
| 10 | Nov |
| 11 | Dec |

## Dictionary

- Dictionaries are good for storing data that can be represented as a two-column table.
- They store unordered collections of key/value pairs.
- A dictionary is enclosed in curly braces, and sets of Key/Value pairs are separated by commas
- A colon is written between each key and value. Commas separate key:value pairs.

```
{ 'TP53' : 'GATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAGTTTTGAGCTTCTCAAAAGTC' ,
'BRCA1' : 'GTACCTTGATTTCGTATTCTGAGAGGCTGCTGCTTAGCGGTAGCCCCTTGGTTTCCGTGGCAACGGAAAA' }
```

| Key | Value |
|---|---|
| TP53 | GATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAGTTTTGAGCTTCTCAAAAGTC |
| BRCA1 | GTACCTTGATTTCGTATTCTGAGAGGCTGCTGCTTAGCGGTAGCCCCTTGGTTTCCGTGGCAACGGAAAA |

## Command line parameters: A Special Built-in List

Command line parameters follow the name of a script or program and have spaces between them. They allow a user to pass information to a script on the command line when that script is being run. Python stores all the pieces of the command line in a special list called `sys.argv`.

You need to import the module named `sys` at the beginning of your script like this

```
!/usr/bin/env python3
import sys
```

Let's imagine we have a script called 'friends.py'. If you write this on the command line:

```
$ friends.py Joe Anita
```

This happens inside the script:

> the script name 'friends.py', and the strings 'Joe' and 'Anita' appear in a list called `sys.argv`.

> These are the command line parameters, or arguments you want to pass to your script.
> `sys.argv[0]` is the script name.
> You can access values of the other parameters by their indices, starting with 1, so `sys.argv[1]` contains 'Joe' and `sys.argv[2]` contains 'Anita'. You access elements in a list by adding square brackets and the numerical index after the name of the list.
> If you wanted to print a message saying these two people are friends, you might write some code like this

```
!/usr/bin/env python3
import sys
friend1 = sys.argv[1] # get first command line parameter
friend2 = sys.argv[2] # get second command line parameter
 now print a message to the screen
print(friend1,'and',friend2,'are friends')
```

The advantage of getting input from the user from the command line is that you can write a script that is general. It can print a message with any input the user provides. This makes it flexible.
The user also supplies all the data the script needs on the command line so the script doesn't have to ask the user to input a name and wait until the user does this. The script can run on its own with no further interaction from the user. This frees the user to work on something else. Very handy!

## What kind of object am I working with?

You have an identifier in your code called `data`. Does it represent a string or a list or a dictionary? Python has a couple of functions that help you figure this out.

| Function | Description |
|---|---|
| `type(data)` | tells you which class your object belongs to |
| `dir(data)` | tells you which methods are available for your object |
| `id(data)` | tells you the unique object id |

We'll cover `dir()` in more detail later

```
>>> data = [2,4,6]
>>> type(data)
<class 'list'>
>>> data = 5
>>> type(data)
<class 'int'>
>>> id(data)
44990666544
```

---

# Link to Python 1 Problem Set

---