Introduction to Python
Part 1

v0.4

Research Computing Services
Information Services & Technology
RCS Team and Expertise

- **Our Team**
  - Scientific Programmers
  - Systems Administrators
  - Graphics/Visualization Specialists
  - Account/Project Managers
  - Special Initiatives (Grants)
  - Maintains and administers the Shared Computing Cluster
    - Located in Holyoke, MA
    - ~19,000 CPUs running Linux

- **Consulting Focus:**
  - Bioinformatics
  - Data Analysis / Statistics
  - Molecular modeling
  - Geographic Information Systems
  - Scientific / Engineering Simulation
  - Visualization

- **CONTACT US:** help@scv.bu.edu
About You

- Working with Python already?
- Have you used any other programming languages?
- Why do you want to learn Python?
Running Python for the Tutorial

- If you have an SCC account, log onto scc4 and use Python there.
  - Run:
    ```bash
    module load python/3.6.2
    spyder &
    /scratch/Intro_Python_code_0.4.sh
    ```
Links on the Rm 107 Terminals

- On the Desktop open the folders:
  Tutorial Files → RCS_Tutorials → Tutorial Files → Introduction to Python

- Copy the whole *Introduction to Python* folder to the desktop or to a flash drive.
  - When you log out the desktop copy will be deleted!
Run Spyder

- Click on the Start Menu in the bottom left corner and type: **spyder**

- After a second or two it will be found. Click to run it.

- Be patient…it takes a while to start.
Running Python: Installing it yourself

- There are **many** ways to install Python on your laptop/PC/etc.
  - [https://www.python.org/downloads/](https://www.python.org/downloads/)
  - [https://www.anaconda.com/download/](https://www.anaconda.com/download/)
  - [https://python-xy.github.io/](https://python-xy.github.io/)
BU’s most popular option: Anaconda

- [https://www.anaconda.com/download/](https://www.anaconda.com/download/)

- Anaconda is a packaged set of programs including the Python language, a huge number of libraries, and several tools.

- These include the **Spyder** development environment and **Jupyter** notebooks.

- Anaconda can be used on the SCC, with some caveats.
Python 2 vs. 3

  - Python 2 is in “maintenance mode” – no new features are expected

- Py3 is not completely compatible with Py2
  - For learning Python these differences are almost negligible

- Which one to learn?
  - If your research group / advisor / boss / friends all use one version that’s probably the best one for you to choose.
  - If you have a compelling reason to focus on one vs the other
  - Otherwise just choose Py3. This is where the language development is happening!
Spyder – a Python development environment

- **Pros:**
  - Faster development
  - Easier debugging!
  - Helps organize code
  - Increased efficiency

- **Cons**
  - Learning curve
  - Can add complexity to smaller problems
Tutorial Outline – Part 1

- What is Python?
- Operators
- Variables
- Functions
- Classes
- If / Else
Tutorial Outline – Part 2

- Lists
- Tuples and dictionaries
- Modules
- numpy and matplotlib modules
- Script setup
- Classes
- Development notes
Tutorial Outline – Part 1

- What is Python?
  - Operators
  - Variables
  - Functions
  - Classes
  - If / Else
What is Python?

- Python…
  - …is a general purpose interpreted programming language.
  - …is a language that supports multiple approaches to software design, principally structured and object-oriented programming.
  - …provides automatic memory management and garbage collection
  - …is extensible
  - …is dynamically typed.

- By the end of the tutorial you will understand all of these terms.
Some History

- “Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas…I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).”

  –Python creator Guido Van Rossum, from the forward to Programming Python (1st ed.)

Goals:

- An easy and intuitive language just as powerful as major competitors
- Open source, so anyone can contribute to its development
- Code that is as understandable as plain English
- Suitability for everyday tasks, allowing for short development times
Compiled Languages (ex. C++ or Fortran)

- **header files**
  - `iostream.h`
  - `my_header.h`
  - `main.cpp`

C++ preprocessor

- *Expanded source code file*
- *not normally visible*
- *`g++ -E` to see output*

C++ compiler

- *Assembler code file*
- *not normally visible*
- *`g++ -S` to see output*

**Object code file**
- `main.o`

Assembler

**C++ library files**
- system library files

**linker**

Executable
- `main`

**`g++ -o main main.cpp`**
Interpreted Languages (ex. Python or R)

- A lot less work is done to get a program to start running compared with compiled languages!
- Python programs start running immediately – no waiting for the compiler to finish.
- Bytecodes are an internal representation of the text program that can be efficiently run by the Python interpreter.
- The interpreter itself is written in C and is a compiled program.
Comparison

Interpreted

- Faster development
- Easier debugging
  - Debugging can stop anywhere, swap in new code, more control over state of program
- (almost always) takes less code to get things done
- Slower programs
  - Sometimes as fast as compiled, rarely faster
- Less control over program behavior

Compiled

- Longer development
  - Edit / compile / test cycle is longer!
- Harder to debug
  - Usually requires a special compilation
- (almost always) takes more code to get things done
- Faster
  - Compiled code runs directly on CPU
  - Compiler can optimize more extensively
- More control over program behavior
The Python Prompt

- The standard Python prompt looks like this:

```
[bgregor@scc2 bg]$ python
Python 3.6.2 (default, Aug 30 2017, 15:46:55)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

- The IPython prompt in Spyder looks like this:

```
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 03:27:45) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.
In [1]:
```

- IPython adds some handy behavior around the standard Python prompt.
The Spyder IDE

- Editor
- Python console
- Variable and file explorer
Tutorial Outline – Part 1

- What is Python?
  - Operators
- Variables
- Functions
- Classes
- If / Else
Operators

- Python supports a wide variety of operators which act like functions, i.e. they do something and return a value:
  - Arithmetic: +, -, *, /, %, **
  - Logical: and, or, not
  - Comparison: >, <, >=, <=, !=, ==
  - Assignment: =
  - Bitwise: & | ~ ^ >> <<
  - Identity: is, is not
  - Membership: in, not in
Try Python as a calculator

- Go to the Python prompt.
- Try out some arithmetic operators:

  +  -  *  /  %  **  ==  ( )

- Can you identify what they all do?
## Try Python as a calculator

- Go to the Python prompt.
- Try out some arithmetic operators:

  \[
  + \quad - \quad * \quad / \quad \% \quad ** \quad == \quad ()
  \]

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division (Note: 3 / 4 is 0.75!)</td>
</tr>
<tr>
<td>%</td>
<td>Remainder (aka <em>modulus</em>)</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>==</td>
<td>Equals</td>
</tr>
</tbody>
</table>
More Operators

- Try some comparisons and Boolean operators. True and False are the keywords indicating those values:

```
In [15]: 4 > 5
Out[15]: False

In [16]: 6 > 3 and 3 > 0
Out[16]: True

In [17]: not False
Out[17]: True

In [18]: True and (False or not False)
Out[18]: True

In [19]:
```
Comments

- `#` is the Python comment character. On any line everything after the `#` character is ignored by Python.

- There is no multi-line comment character as in C or C++.

- An editor like Spyder makes it very easy to comment blocks of code or vice-versa. Check the *Edit* menu.
Tutorial Outline – Part 1

- What is Python?
- Operators
- Variables
- Functions
- Classes
- If / Else
Variables

- Variables are assigned values using the = operator.
- In the Python console, typing the name of a variable prints its value.
  - Not true in a script!
- Variables can be reassigned at any time.
- Variable type is not specified.
- Types can be changed with a reassignment.
Variables cont’d

- Variables refer to a value stored in memory and are created when first assigned.
- Variable names:
  - Must begin with a letter (a - z, A - Z) or underscore _
  - Other characters can be letters, numbers or _
  - Are case sensitive: capitalization counts!
  - Can be any reasonable length
- Assignment can be done *en masse*:
  \[
  x = y = z = 1
  \]
- Multiple assignments can be done on one line:
  \[
  x, y, z = 1, 2.39, 'cat'
  \]

Try these out!
Variable Data Types

- Python determines data types for variables based on the context.

- The type is identified when the program runs, called **dynamic typing**.
  - Compare with compiled languages like C++ or Fortran, where types are identified by the programmer and by the compiler **before** the program is run.

- Run-time typing is very convenient and helps with rapid code development…but requires the programmer to do more code testing for reliability.
  - The larger the program, the more significant the burden this is!
Variable Data Types

- Available basic types:
  - **Numbers**: Integers and floating point (64-bit)
  - **Complex numbers**: \( x = \text{complex}(3, 1) \) or \( x = 3 + 1j \)
  - **Strings**, using double or single quotes: "cat" and 'dog'
  - **Boolean**: `True` and `False`
  - **Lists, dictionaries, and tuples**
    - These hold collections of variables
    - Specialty types: files, network connections, objects
  
- Custom types can be defined using Python classes.
Variable modifying operators

- Some additional arithmetic operators that modify variable values:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Effect</th>
<th>Equivalent to…</th>
</tr>
</thead>
<tbody>
<tr>
<td>x += y</td>
<td>Add the value of y to x</td>
<td>x = x + y</td>
</tr>
<tr>
<td>x -= y</td>
<td>Subtract the value of y from x</td>
<td>x = x - y</td>
</tr>
<tr>
<td>x *= y</td>
<td>Multiply the value of x by y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>x /= y</td>
<td>Divide the value of x by y</td>
<td>x = x / y</td>
</tr>
</tbody>
</table>

- The += operator is by far the most commonly used of these.
Check a type

- A built-in function, `type()`, returns the type of the data assigned to a variable.
  - It’s unusual to need to use this in a program, but it’s available if you need it!

- Try this out in Python – do some assignments and reassignments and see what `type()` returns.
Strings

- Strings are a basic data type in Python.

- Indicated using pairs of single " or double "" quotes.

- Multiline strings use a triple set of quotes (single or double) to start and end them.

```python
'cat'

"dog"

"What's that?"

'"They said "hello""'

'"'"This is a multiline string "'"'
```
Tutorial Outline – Part 1

- What is Python?
- Operators
- Variables
- Functions
- Classes
- If / Else
Functions

- Functions are used to create code that can be used in a program or in other programs.

- The use of functions to logically separate the program into discrete computational steps.

- Programs that make heavy use of function definitions tend to be easier to develop, debug, maintain, and understand.
Python functions

- The return value can be any Python type
- If the return statement is omitted a special `None` value is still returned.
- The arguments are optional but the parentheses are required!
- Functions must be defined before they can be called.
Function Return Values

- A function can return any Python value.

- Function call syntax:

```python
A = some_func()  # some_func returns a value
Another_func()   # ignore return value or nothing returned
b,c = multiple_vals(x,y,z)  # return multiple values
```

- Open `function_calls.py` for some examples
Function arguments

- Function arguments can be required or optional.
- Optional arguments are given a default value

```python
def my_func(a, b, c=10, d=-1):
    ...some code...
```

- To call a function with optional arguments:
- Optional arguments can be used in the order they’re declared or out of order if their name is used.

```python
my_func(x, y, z)    # a=x, b=y, c=z, d=-1
my_func(x, y)      # a=x, b=y, c=10, d=-1
my_func(x, y, d=w, c=z)  # a=x, b=y, c=z, d=w
```
Garbage collection

- Variables defined in a function (or in any code block) no longer have any “live” references to them once the function returns.

- These variables become garbage

- Python’s garbage collection operates to remove them from the computer’s memory, freeing up the memory to be re-used.

- There is no need to explicitly destroy or release most variables.
  - Some complex data types provide .close(), .clean(), etc. type functions. Use these where available.
When does garbage collection occur?

- It happens when Python thinks it should.
- For the great majority of programs this is not an issue.
- Programs using very large quantities of memory or allocating large chunks of memory in repeated function calls can use more memory than expected.
- If you are having trouble with memory usage contact RCS for help!
Tutorial Outline – Part 1

- What is Python?
- Operators
- Variables
- Functions
- Classes
- If / Else
Classes

- **OOP:** Object Oriented Programming

- In OOP a *class* is a data structure that combines data with functions that operate on that data.

- An *object* is a variable whose type is a *class*
  - Also called an *instance* of a class

- Classes provide a lot of power to help organize a program and can improve your ability to re-use your own code.
Object-oriented programming

- Python is a fully object oriented programming (OOP) language.

- Object-oriented programming (OOP) seeks to define a program in terms of the *things* in the problem (files, molecules, buildings, cars, people, etc.), what they need, and what they can do.

- Some familiarity with OOP is needed to understand Python data structures and libraries.

- You can write your own Python classes to define custom data types.
Object-oriented programming

- OOP defines classes to represent the parts of the program.
- Classes can contain data and methods (internal functions).
- Classes can inherit from one another
  - A class (the subclass) can use all of the data and methods from another class (the superclass) and add its own.
- This is a highly effective way of modeling real world problems inside of a computer program.
Classes bundle data and functions

- In Python, calculate the area of some shapes after defining some functions.

```python
radius = 14.0
width_square = 14.0
a1 = AreaOfCircle(radius)  # ok
a2 = AreaOfSquare(width_square)  # ok
a3 = AreaOfCircle(width_square)  # !! OOPS
```

- If we defined Circle and Rectangle classes with their own area() methods... it is not possible to miscalculate.

```python
# create a Circle object with the radius value
c1 = Circle(radius)
r1 = Square(width_square)

a1 = c1.area()
a2 = r1.area()
```
Strings in Python

- Python defines a string class – all strings in Python are objects.

- This means strings have:
  - Their own internal (hidden) memory management to handle storage of the characters.
  - A variety of functions accessible once you have a string object in memory.

- You can’t access string functions without a string – in Python the string provides its own functions.
  - No “strcat” / “strcmp” / etc as in C
  - No “strlen” / “isletter” / etc as in Matlab
  - No “nchar” / “toupper” /etc as in R
String functions

- In the Python console, create a string variable called `mystr`

- type: `dir(mystr)`

- Try out some functions:
  
  - `mystr.upper()`
  - `mystr.title()`
  - `mystr.isdecimal()`

- Need help? Try:
  
  `help(mystr.title)`

```python
mystr = 'Hello!
```
The `len()` function

- The `len()` function is not a string specific function.
- It’ll return the length of any Python object that contains any countable thing. `len(mystr) \rightarrow 6`
- In the case of strings it is the number of characters in the string.
String operators

- Try using the + and += operators with strings in the Python console.

- + concatenates strings.
- += appends strings.
  - These are defined in the string class as functions that operate on strings.

- Index strings using square brackets, starting at 0.
String operators

- Changing elements of a string by an index is **not allowed**:

  ```python
  In [79]: a='Hello BU!!'
  In [80]: a[4] = '0'
  Traceback (most recent call last):
      File "<ipython-input-80-7c5733c2cb67>", line 1, in <module>
        a[4] = '0'
  TypeError: 'str' object does not support item assignment
  ```

- Python strings are **immutable**, i.e. they can't be changed.
String Substitutions

- Python provides an easy way to stick variable values into strings called substitutions.

- Syntax for one variable:

```
'string with a %s' % variable
```

- For more than one:

```
'x: %s  y: %s  z: %s' % (xval,yval,zval)
```

- Printing:

```
print('x: %s, y: %s, z:%s' % (xval,yval,2.0))
```
Tutorial Outline – Part 1

- What is Python?
- Operators
- Variables
- Functions
- Classes
- If / Else
If / Else

- *If*, *elif*, and *else* statements are used to implement conditional program behavior.
- **Syntax:**

```python
if Boolean_value:
    ...some code
elif Boolean_value:
    ...some other code
else:
    ...more code
```

- *elif* and *else* are not required – use them to chain together multiple conditional statements or provide a default case.
• Try out something like this in the Spyder editor.

• Do you get any error messages in the console?

• Try using an *elif* or *else* statement by itself without a preceding *if*. What error message comes up?
Which code sample is easier to read?

- C:

```c
int x;
if (3 > 4) {
    x = 5;
} else {
    x = 6;
}
```

- Matlab:

```matlab
if (3 > 4)
    x = 5
else
    x = 6
end
```

or

```c
int x;
if (3 > 4) {
    x = 5;
} else {
    x = 6;
}
```

or

```matlab
if (3 > 4)
    x = 5
else
    x = 6
end
```
Most languages use special characters ({ } pairs) or keywords (end, endif) to indicate sections of code that belong to:

- Functions
- Control statements like if
- Loops like for or while

Python instead uses the indentation that programmers use anyway for readability.
The Use of Indentation

- Python uses whitespace (spaces or tabs) to define *code blocks*.
- Code blocks are logical groupings of commands. They are *always* preceded by a colon :

```
if 3 > 4:
    x = 5
else:
    x = 6
```

- This pattern is consistently repeated throughout Python syntax.
- Spaces or tabs can be mixed in a file but *not* within a code block.
If / Else code blocks

- Python knows a code block has ended when the indentation is removed.

- Code blocks can be nested inside others therefore *if-elif-else* statements can be freely nested within others.

```python
a = 1
b = 2
if a <= b:
    c = a
    print('a <= b')
    if c == 1:
        print('c is 1')
print('out of the if statement')
```
File vs. Console Code Blocks

- Python knows a code block has ended when the indentation is removed.

- EXCEPT when typing code into the Python console. There an empty line indicates the end of a code block.

- This sometimes causes confusion when pasting code into the console.
  - Solution: try to avoid pasting more than a few lines into the Python console.