Crash Course in Python; Unit 0

Introduction to Python Programming

UNIVERSITY OF CALGARY

Learning Outcomes

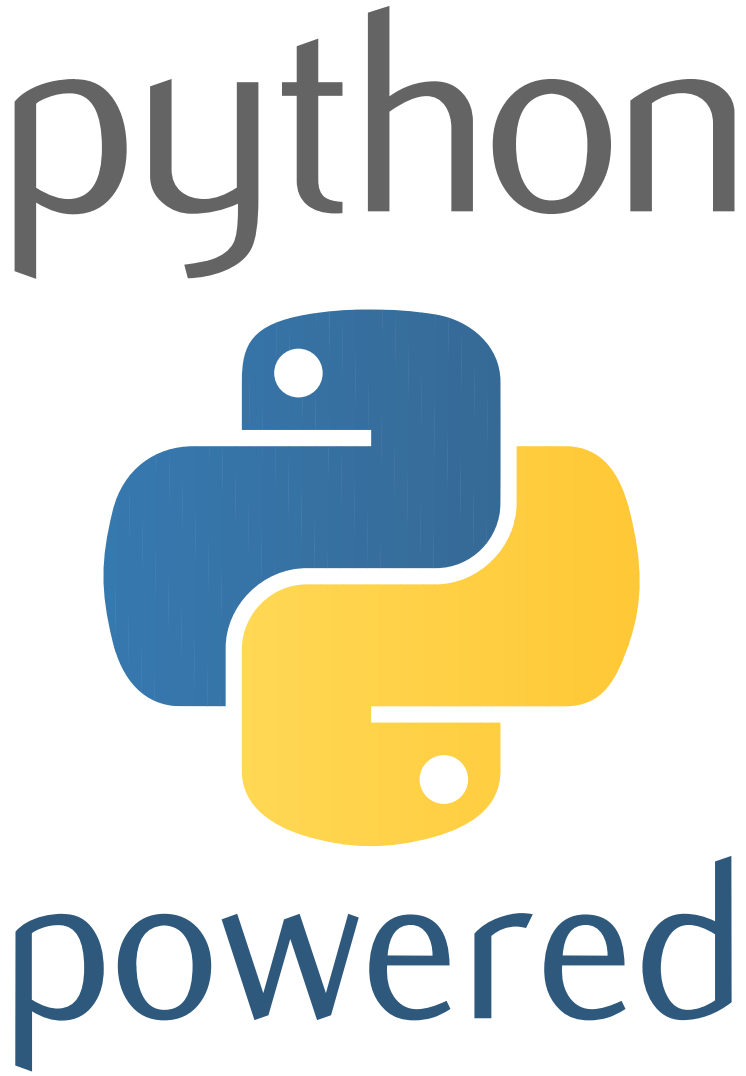
By the end of the module, you will be able to:

- Run Python scripts on your computer
- Create a custom Python script which can:
  - Request and store user input
  - Compare values to one another
  - Do simple arithmetic on numerical values
  - Report the results of your process to the user

All Python code described and used in this slideshow will be provided in a Jupyter Notebook file, which you can run on TALC to further experiment with. A (non-runnable) PDF version of said code will also be available for reference.

What is Python?

- Initially created in 1990, and named in homage to Monty Python
- A high-level programming language made to automate the details of computer programming for the user
- Built with a focus on being easy to use, portable, and easily extendable through packages
- Very common in nearly every modern computer application, including website management, data science, and cryptography.



https://www.python.org/static/community_logos/python-powered-h.svg

Syntax Used in These Lectures

- Shell command (`cmd` on Windows, `bash` for Linux/MacOS)

```
$ {command}
```

- Python command

```
~ {command}
```

- Expected command output

```
> {output}
```

Checking Local Python Version

- Most modern operating systems will already have Python installed; this can be checked with the following command on your OS's command line:

```
$ python --version
# Python 3.8.5
```

- Sometimes the 'py' command needs to used instead:

```
$ py --version
# Python 3.8.5
```

- If no version of Python is installed, please follow the instructions at python.org/downloads/ to install Python 3.8.x or above.

Running Python Interactively

- Running the ‘python’ or ‘py’ command (as you did in the prior slide) without any flags will put the console into interactive Python mode:

```
$ python
~
```

  - Note: On Windows you may need to open a Python console manually instead (like any other program)

- When in this mode, you can run Python code as if it were within a script file. For example, let’s print out “Hello World!” to the console:

```
~ print("Hello World!")
> ‘Hello World!’
```

- To end the interactive Python session, simply type “exit()” and hit enter:

```
~ exit()
$
```

Common Python Utilities

UNIVERSITY OF CALGARY

The ‘print’ Function

- Prints out what is provided to it, allowing it to be viewed by the user
- Extremely helpful for keeping track of a program’s state as it runs, as well as for debugging purposes
- A simple example, with a sentence:

```
~ print("I love Python!")
> ‘I love Python!’
```

- Another example, with a number:

```
~ print(42)
> 42
```

https://fthmb.tqn.com/N8Uimk910SVcRhkFx2bJu5bzGy4=/1732x1732/filters:fill(auto,1)/GettyImages-542929476-5935e3373df78c08abe910c2.jpg

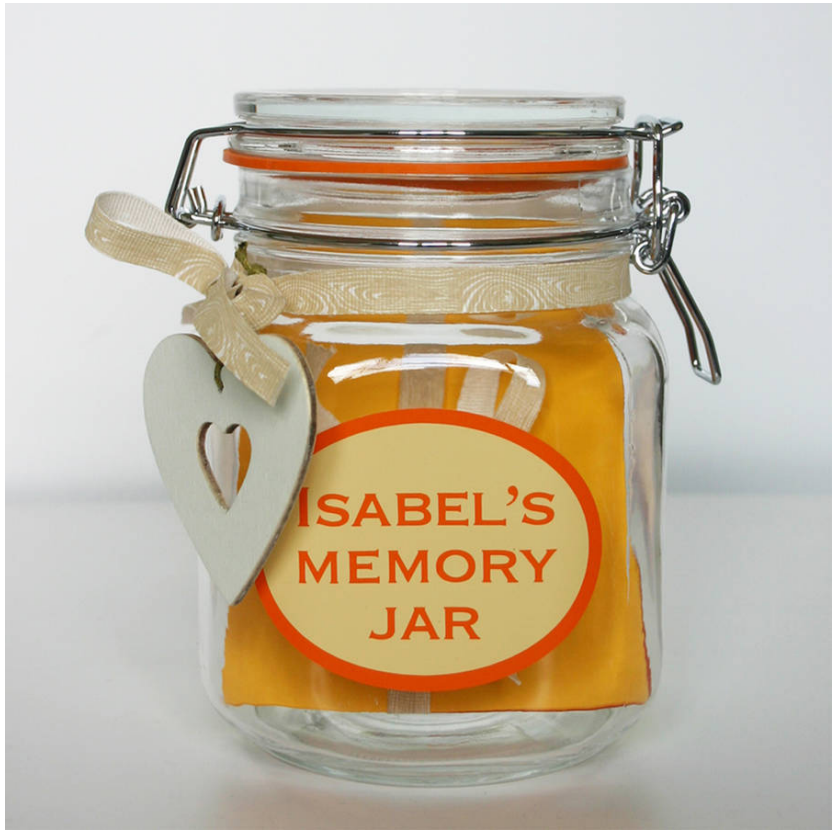
Common Python Utilities; Variable Creation

- Variables are named containers which store values within Python
- Can be created using the '=', with the variable name on the left, and the value to assign to it on the right:

```
~ foo = "bar"
```

- The contents of a variable can then be printed out using the 'print' function

```
~ print(foo)
> 'bar'
```



https://cdn.notonthehighstreet.com/system/product_images/images/002/156/670/original_personalised-memory-jar.jpg

Common Python Utilities; Variable Modification

- Existing variables can have their contents overridden in the same way:

```
~ foo = "bar"
~ print(foo)
> ‘bar’
~ foo = "boz"
~ print(foo)
> ‘boz’
```

https://cdn.shopify.com/s/files/1/1192/8140/products/Order-icon_1024x1024.png?v=1485845001

Common Python Utilities; Variable Deletion

- We can also delete variables we no longer need with the “del” statement:

```
~ foo = "bing"
~ print(foo)
> ‘bing’
~ del foo
~ print(foo)
> Traceback (most recent call last):
>   File "<stdin>", line 1, in <module>
> NameError: name 'foo' is not defined
```

https://webstockreview.net/images/can-clipart-png-18.png

Common Python Utilities; Variable Naming

- Variable names must abide by the following rules in Python:
  - Must not contain any symbols, with the exception of “_”
  - Cannot start with a number (0-9)
  - Cannot be identical to a name already built into Python (i.e. “print”)
- Variables are case sensitive; uppercase “Foo” can contain a different value than lowercase “foo”
- All variables can store any type of data at any time (numeric, text, boolean etc.). It is generally good practice to stick to one type per variable whenever possible, however

Common Python Utilities; Introduction to Operators

- Operators are built in symbols which, when provided with some values, will modify them in some way
- We've already used one such operator; the equal sign (=), which is the 'assignment' operator. It assigns the value on it's right to the variable on its left:

```
~ foo = "bar"
~ print(foo)
$ 'bar'
~ bing = "fiz"
~ print(bing)
$ 'fiz'
~ foo = bing
~ print(foo)
$ 'fiz'
```

Common Python Utilities; Arithmetic Operators

- A set of operators which perform basic arithmetic on numeric values
- Will provide the result of their operation immediately after completion:

```
~ 2+3
> 5
```

- The results of these can be stored in variables for later use:

```
~ tmp_prod = 3*7
~ print(tmp_prod)
> 21
```

Common Python Utilities; Arithmetic Operator Table

		Operator						
Left Value	**Right Value**	**Add +**	**Subtract -**	**Multiply ***	**Divide /**	**Modulus %**	**Floor //**	**Power ****
4	**2**	6	2	8	2	0	4	16
1	**3**	4	-2	3	0.333333	1	0	1
5	**3**	8	2	15	1.666667	2	3	125
3	**0**	3	3	0	*ERROR*	*ERROR*	*ERROR*	1

Common Python Utilities; Assignment Operators

- These operators will update the contents of a provided variable
- The variable to be updated needs to be on the left, and the updating value (which itself can be a variable or use operators) on the right:

```
~ foo = 7+8
~ print(foo)
> 15
```

- They will often use the initial value of the variable as part of their operation

```
~ tmp_prod = 4
~ tmp_prod *= 3
~ print(tmp_prod)
> 12
```

Common Python Utilities; Assignment Operator Table

		Operator							
Initial Value	**Assigned Value**	**Assign =**	**Sum +=**	**Delta -=**	**Product *=**	**Quotient /=**	**Modulo %=**	**Floor Quotient //=**	**Exponent **=**
4	**2**	2	6	2	8	2	0	4	16
1	**3**	3	4	-2	3	0.333333	1	0	1
5	**3**	3	8	2	15	1.666667	2	3	125
3	**0**	0	3	3	0	*ERROR*	*ERROR*	*ERROR*	1

Common Python Utilities; Comparison Operators

- Compare two values to one another, evaluating as ‘True’ if the operator’s condition is met, and as ‘False’ otherwise

```
~ foo = 4
~ bar = 5
~ foo<bar
> True
```

- The result of a comparison can be stored for later use, like any other value type

```
~ bing = "Hello"=="World"
~ print(bing)
> False
```

Common Python Utilities; Comparison Operator Table

		Operator					
Left Value	**Right Value**	**Equal ==**	**Not Equal !=**	**Greater Than >**	**Lesser Than <**	**Greater or Equal to >=**	**Less or Equal to >=**
4	**2**	False	True	True	False	True	False
2	**4**	False	True	False	True	False	True
3	**3**	True	False	False	False	True	True

Common Python Utilities; Logical Operators

- Allow for common logical checks between two boolean (true/false) values
- Provides a “True” or “False” result, depending on whether the logical check succeeds

```
~ foo = True
~ bar = False
~ foo and bar
> False
```

- Can be grouped using brackets to allow for more complex comparisons

```
~ bing = (420==69 and 34<43) or "Apple"!="Orange"
~ print(bing)
> True
```

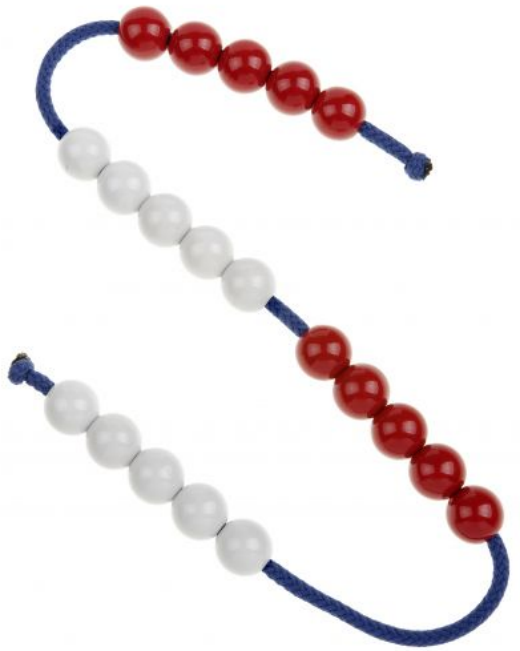
Common Python Utilities; Logical Operator Table

		Operator		
Left Value	**Right Value**	**not (left only)**	**and**	**or**
TRUE	**TRUE**	FALSE	TRUE	TRUE
TRUE	**FALSE**	FALSE	TRUE	TRUE
FALSE	**FALSE**	TRUE	FALSE	FALSE

Common Python Utilities; Strings

- A special type of data in Python, acting as a collection of alphanumeric characters ‘strung’ together (hence the name)
- Can be created using single or double quotes:

```
~ str1 = "Hello "
~ str2 = ‘World!’
~ print(str1 + str2)
> ‘Hello World!’
```



https://www.123learning.co.uk/assets/galleries/257/new_image_-_20_beads.jpg

Common Python Utilities; String Concatenation

- Two strings can be “added” to one another using the + and += operators:

```
~ str1 = "Foo"
~ str2 = "Bar"
~ str3 = "Baz"
~ str4 = str1 + str2
~ str4 += str3
~ print(str4)
> ‘FooBarBaz’
```

UNIVERSITY OF CALGARY

http://s3-external-1.amazonaws.com/wootdesigncontestentries/brxan/Insert_Funny_Text_Here-bpowec-s.jpg

Common Python Utilities; String Conversion

- We can convert non-string types into strings using the ‘str’ command:

```
~ num_str = "Val" + str(43)
~ print(num_str)
> ‘Val43’
```

- This can be done implicitly within a larger string of text using f-strings (created by leading the initial quote with an ‘f’) and curly brackets ‘{}’

```
~ a = 6
~ b = 8
~ sentence = f"The sum of {a} and {b} is {a+b}"
~ print(sentence)
> ‘The sum of 6 and 8 is 14’
```

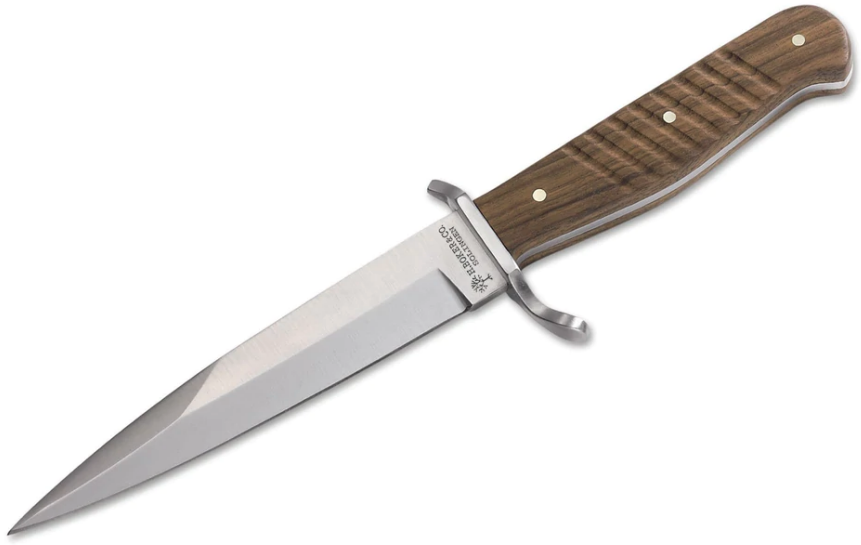
Common Python Utilities; String Slicing, Part 1

- We can select parts of strings (slicing them) using square brackets '[ ]' and colons ':'.
- Consider the string "`I love Python!`", saved in the variable "`python_love`":
  - Single character (*note; the first element starts at 0!*):

```
~ i1 = print(python_love[0])
~ print(i1)
> 'I'
```

  - All characters past index (inclusive)

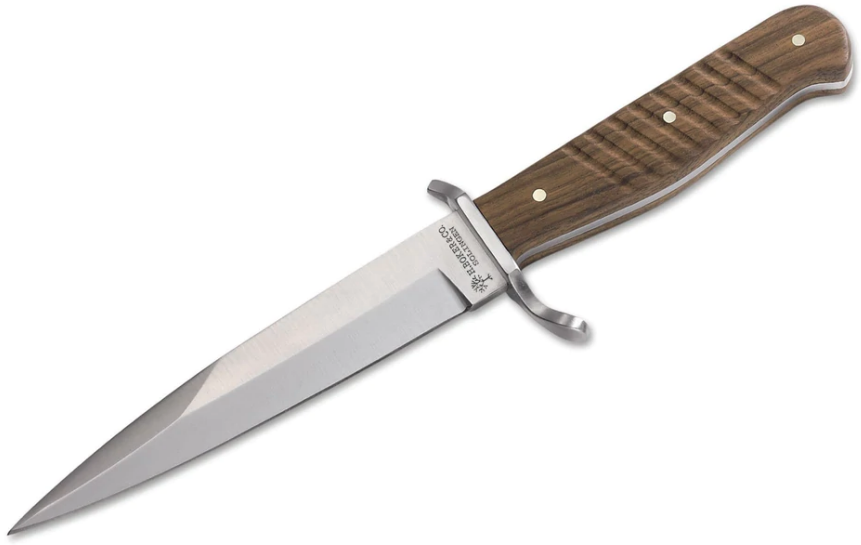
```
~ s1 = python_love[7:]
~ print(s1)
> 'Python!'
```



https://cdn.shopify.com/s/files/1/0004/8893/1386/products/5852b9b6476cf8.89329607_1024x1024.jpg?v=1575933972

Common Python Utilities; String Slicing, Part 2

- Consider the string "`I love Python!`", saved in the variable "`python_love`":
  - All characters before index (exclusive)
    - `s2 = python_love[:7]`
    - `print(s2)`
    - `> 'I love '`
  - All characters between two indices (inclusive/exlcusive)
    - `s3 = python_love[2:6]`
    - `print(s3)`
    - `> 'love'`



https://cdn.shopify.com/s/files/1/0004/8893/1386/products/5852b9b6476cf8.89329607_1024x1024.jpg?v=1575933972

Common Python Utilities; Requesting User Input

- We can prompt the user to provide input with the '`input`' command. All this command requires to function is a string to prompt the user with:

```
~ user_input = input("Please enter a name:")
> Please enter a name: Joe
~ print(f"Hello {user_input}!")
> Hello Joe!
```

- Any input captured in this form is treated as a string by default:

```
~ a = input("Please enter a number:")
> Please enter a number: 3
~ b = input("Please enter a second number:")
> Please enter a second number: 5
~ print(a + b)
> '35'
```

Common Python Utilities; Type Conversion

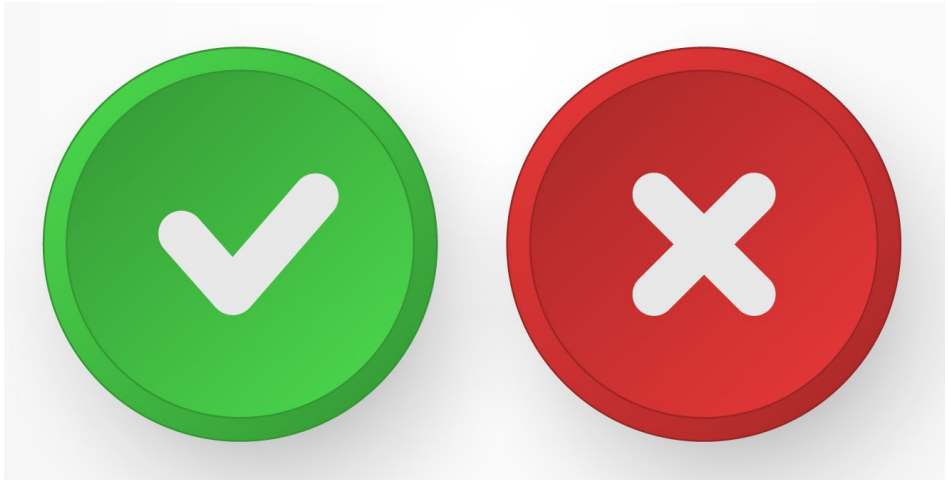
- Similar to how elements can be converted to strings, we can attempt to convert strings back into a different type using the '`int`' (integers) or '`float`' (decimal) commands

```
> a = input("Enter an Integer: ")
~ Enter and Integer: 4
> print(a + 4)
~ Traceback (most recent call last):
~   File "<stdin>", line 1, in <module>
~ TypeError: can only concatenate str (not "int") to str
> print(int(a) + 4)
~ 8
> print(float(a) + 4)
~ 8.0
```

Common Python Utilities; Boolean Conversion

- The ‘`bool`’ command can be used to request python to try and convert a string into it’s True/False equivalent; the resulting value can be used with Logical operators.

```
> a = bool(input("Are you paying attention? (yes/no)"))
~ Are you paying attention? (yes/no) yes
> print(f"It is {a} that you’ll do well!")
~ ‘It is True that you’ll do well in this course’
```



https://clipground.com/images/yes-no-clipart-free-9.jpg

Common Python Utilities; Comments

- We can add remarks to Python code using the '#' symbol. Any text behind the first hashtag will not be run, but will remain in-code for our (and other people's) reference

```
~ # Pythagorean theorem demo
~ a = 3
~ b = 4
~ c = (a**2 + b**2)**(0.5)
~ print(c)
> 5.0
```

https://cdn.pngsumo.com/remark-icons-download-426-free-png-and-vector-icons-unlimited-remark-png-512_512.png

Putting it All Together!

UNIVERSITY OF CALGARY

Lets Test your Learning!

- You have been tasked to create a Python program which does the following:
  - Greet the user
  - Request the user to provide their name and two numbers
  - Report if the first number they provided is greater than the second
  - Calculates the sum, difference, product, and quotient (alongside its remainder) of the numbers provided, reporting the results to the user
  - Gives a short farewell to the user, using their name in the process.

An incomplete Python script file has been provided with comment hints for each of the above steps for you to test your learning. A key for this document has also been provided, should you want to check your answer

Thank you!

Please send inquiries to kalum.ost@ucalgary.ca

UNIVERSITY OF CALGARY