Graduate Professional Development Conference

Python Skills for Graduate Students

Dr. Michael Liut

michael.liut@utoronto.ca
Mathematical and Computational Sciences
University of Toronto Mississauga

October 15, 2019

UNIVERSITY OF
TORONTO
MISSISSAUGA

Pre-work: Install Python & Verify Install

For those of you who want to use your personal machines please be sure to install Python 3.7 and the relevant libraries ***before attending*** *the session.*

Task 1

Install Python 3.7.

Task 2

Open Idle (the icon looks like: ) where you should see `>>>_`

Task 3

Type `2+2` then ⏎ (the return/enter key). You should get `4` back.

Pre-work: Install & Check Libraries

Task 4

Install the following libraries: matplotlib, numpy, and pandas.

Task 5

In Idle type `import matplotlib` then ⏎ (the return/enter key). You should see `>>>_` meaning that the library is installed. If there is an error, please go back to Task 4.

Task 6

In Idle type `import numpy` then ⏎ (the return/enter key). You should see `>>>_` meaning that the library is installed. If there is an error, please go back to Task 4.

Pre-work: Check Libraries

Task 7

In Idle type `import pandas` then ↵ (the return/enter key). You should see `>>>_` meaning that the library is installed. If there is an error, please go back to Task 4.

Congrats! You're ready for tomorrow!

Workshop Topics

1. Introduction to Python
2. CSV Manipulation
3. Data Analysis
4. Graphing Data

Python as a Calculator

Standard Arithmetic

addition	subtraction	multiplication	division
+	-	*	/

Other Operations

integer division	exponentiation	modulus
//	**	%

Python provides a `max` and `min` function that, given a sequence of numbers, returns the largest or smallest number:

```
1 >>> min(1,2,3)
2 1
3 >>> max(1,2,3)
4 3
```

Variables

Sometimes it is expedient to give values a name. These names are called variables. For instance:

```
1 >>> x = -2
2 >>> y = 3
3 >>> q = x // y
4 >>> r = x % y
5 >>> x - (q * y + r)
6 0
```

Equal in Python is for assignment and does not denote equality as it does in mathematics.

```
1 >>> x = 1
2 >>> x = x + 1
3 >>> x
4 2
```

Notice in mathematics that

$$x = x + 1 \implies 0 = 1$$

and so is an "illegal" statement.

```
>>> 1 = 1
    1 = 1
        ^
SyntaxError: can't assign to literal

>>> 1 == 1
True

>>> x = y
NameError: name 'y' is not defined
```

Functions

Unlike variables, Python functions are directly analogous to that of mathematics and can also be named.

For instance, a parabola in mathematics is given by the function

$$f(x) = x^2$$

and in Python we write

```
1 >>> def f(x):
2 ...     return x**2
3 >>> f(3)
4 9
```

In Python four spaces/indents are significant and are used to associate lines of code with control structures (in this case function definition).

```
1 >>> def␣f(x):
2 ...␣␣␣␣return␣x**2
```

If you get errors along the lines of

```
1 IndentationError: unexpected indent
```

check your formatting.

Return Statement

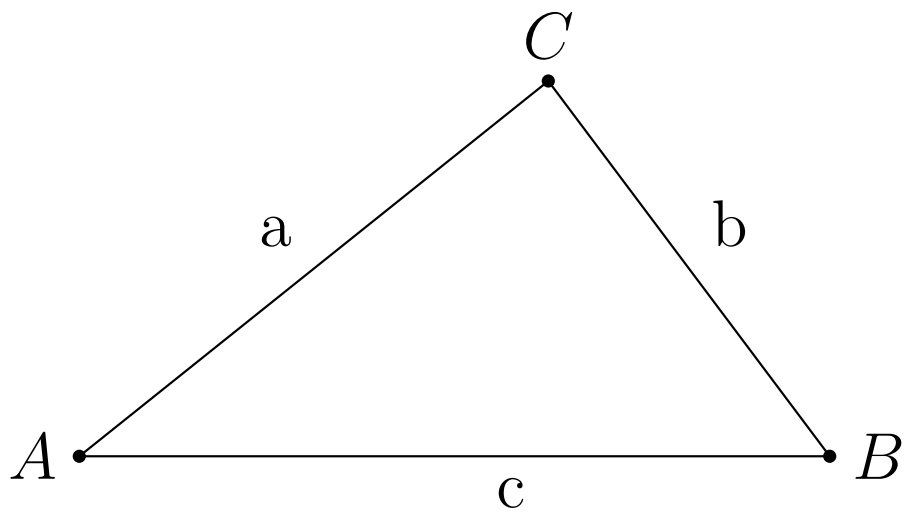
`return` is a reserved word (one that cannot be assigned by us). It is used to designate what value the function should return while also terminating the function itself.

Definition 1 (`return`)

The `return` statement causes a function to exit and hand back a value to its caller.

Example: Area of Triangle

The area of $\triangle ABC$ given by



is $\sqrt{s(s-a)(s-b)(s-c)}$ where $s = \frac{1}{2}(a+b+c)$.

Write a function which computes the area of a triangle from the side lengths a, b, c.

Solution: Area of Triangle

In mathematics we would define a mapping

$$\text{triangle\_area} : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}.$$

In Python we can write:

```
1 >>> def triangle_area(a:int, b:int, c:int) -> float:
2 ...     s = a+b+c
3 ...     s = s/2
4 ...     return (s*(s-a)*(s-b)*(s-c))**0.5
5  though no actual type-checking is performed automatically
6 >>> triangle_area(1.1, 2.2, 3.3)
7 1.194989539703172
```

Logical Operations and Constants

The following are the basic operands of logic:

1. `True`
2. `False`
3. `or`
4. `and`
5. `not`

Definition 2 (Boolean doman)

Let $\mathbb{B}$ denote the boolean domain where

$$\mathbb{B} = \{ \texttt{True} , \texttt{False} \} .$$

Definition 3 (Predicate)

Any function that maps into $\mathbb{B}$ is called a predicate. (i.e. any function that evaluates to `True` or `False` .)

Definition 4 (String)

Anything (with some exceptions) enclosed by single-quotes ‘ ’ or double-quotes “ ” is considered a string by Python.

A string is an ordered collection of the characters (e.g. unicode and ascii) allowed by the computer.

```
>>> "hello world"
'hello world'

>>> type("hello world")
<class 'str'>

>>> hello world                    note the lack of quotes
SyntaxError: invalid syntax

>>> hello                          note the lack of quotes
NameError: name 'hello' is not defined
```

Adding Strings

```
1 >>> "hello" + "world"
2 'helloworld'
3
4 >>> type(" ")
5 <class 'str'>
6
7 >>> space = " "
8 >>> "hello" + space + "world"
9 'hello world'
```

Note when strings are added a new string is created.

String Equality

```
1  >>> "hello" == "hello"
2  True
3
4  >>> "hello " == "hello"
5  False
6
7  >>> "h e l l o" == "hello"
8  False
9
10 >>> "Hello" == "hello"
11 False  strings are case sensitive:  "H" != "h"
```

Comparing Strings

```
1 >>> "a" < "b"
2 True
3
4 >>> "A" < "a"
5 True
6
7 >>> "Z" < "a"
8 True
```

```
>>> "a" < "aa"
True

>>> "b" < "aa"
False

>>> "aba" < "ab"
False

>>> "aZ" < "aa"
True
```

New Line

A new line is an escape character that can be used in strings to print what is subsequent to it on a new line. The new line escape character is \n.

```
1 >>> "hello\nworld"
2 'hello\nworld'
3
4 >>> print("hello\nworld")
5 hello
6 world
```

Notice how a string can be stored differently than it is printed.

Tab

A tab is a fixed amount of horizontal space. How a tab is displayed depends on the program displaying it.

(This is why tabs are the worst :)

```
1 >>> "hello\tworld"
2 'hello\tworld'
3
4 >>> print("hello\tworld")
5 'hello   world'
```

Numbers versus Strings

```
1  >>> 3 + 7
2  10
3  >>> "3" + "7"
4  37
5  >>> 3 + "7"
6  TypeError: unsupported operand type(s) for +: 'int'
7   and 'str'
8  >>> str(3) + "7"
9  37
10 >>> 3 + int("7")
11 10
```

```
>>> 3 + int("7")
10

>>> float("123.456")
123.456
```

This is only true for numbers!

```
>>> int("hello")
ValueError: invalid literal for int() with base 10:
'hello'
```

Substitution

There is a mechanism for printing string variables in sentences through substitution.

```
1 >>> x = "hello"
2 >>> y = "world"
3 >>> z = "{}ooo {}ddd".format(x,y)
4 >>> print(z)
5 helloooo worldddd
```

Length

A strings length is the number of characters that comprise it.

```
1 >>> len("h")
2 1
3 >>> len("hello")
4 5
5 >>> x = "world"
6 >>> len(x)
7 5
8 >>> len(x+"world") == len(x) + len("world")
9 True
```

Inclusion

As a string can be regarded as an ordered set we can use the element of.

```
>>> "h" in "hello world"
True
>>> "hello" in "hello world"
True
>>> x = "world"
>>> x in "hello world"
True
>>> "ow" in "hello world"
False
```

String Indexing

Because a string is ordered we can number its characters starting from zero and access them by using square brackets.

```
1 >>> x = "hello world"
2 >>> x[0]
3 'h'
4 >>> x[1]
5 'e'
6 >>> x[2]
7 'l'
8 >>> x[len(x)]
9 IndexError: string index out of range
```

We can also index from the end.

```
>>> x = "hello world"
>>> x[-1]
'd'
>>> x[-2]
'l'
>>> x[-3]
'r'
```

String Slicing

Because the string's characters are numbered we can slice the string to obtain only a part of it.

```
1 >>> x = "0123456789"                    So index matches character.
2 >>> x[1:4]       grab 1st inclusive through 4th exclusive characters
3 '123'
4 >>> x[0:9]
5 '012345678'
6 >>> x[0:10]
7 '0123456789'
```

Built-In String Method

There are a myriad of built-in string methods in Python. You can review them by searching the web or doing

```
>>> help(str)
 ...
 |  capitalize(self, /)
 |      Return a capitalized version of the string.
 |
 |  casefold(self, /)
 |      Return a version of the string suitable for
 |      caseless comparisons.
```

Type ⏎ to scroll and q to escape or quit help.

Methods

We will eventually learn that strings are objects.

Objects have methods which are like functions but have different invocation syntax.

For instance we do not say

```
>>> capitalize("hello")
NameError: name 'capitalize' is not defined
```

but rather

```
>>> "hello".capitalize()                    Note the ()
'Hello'
```

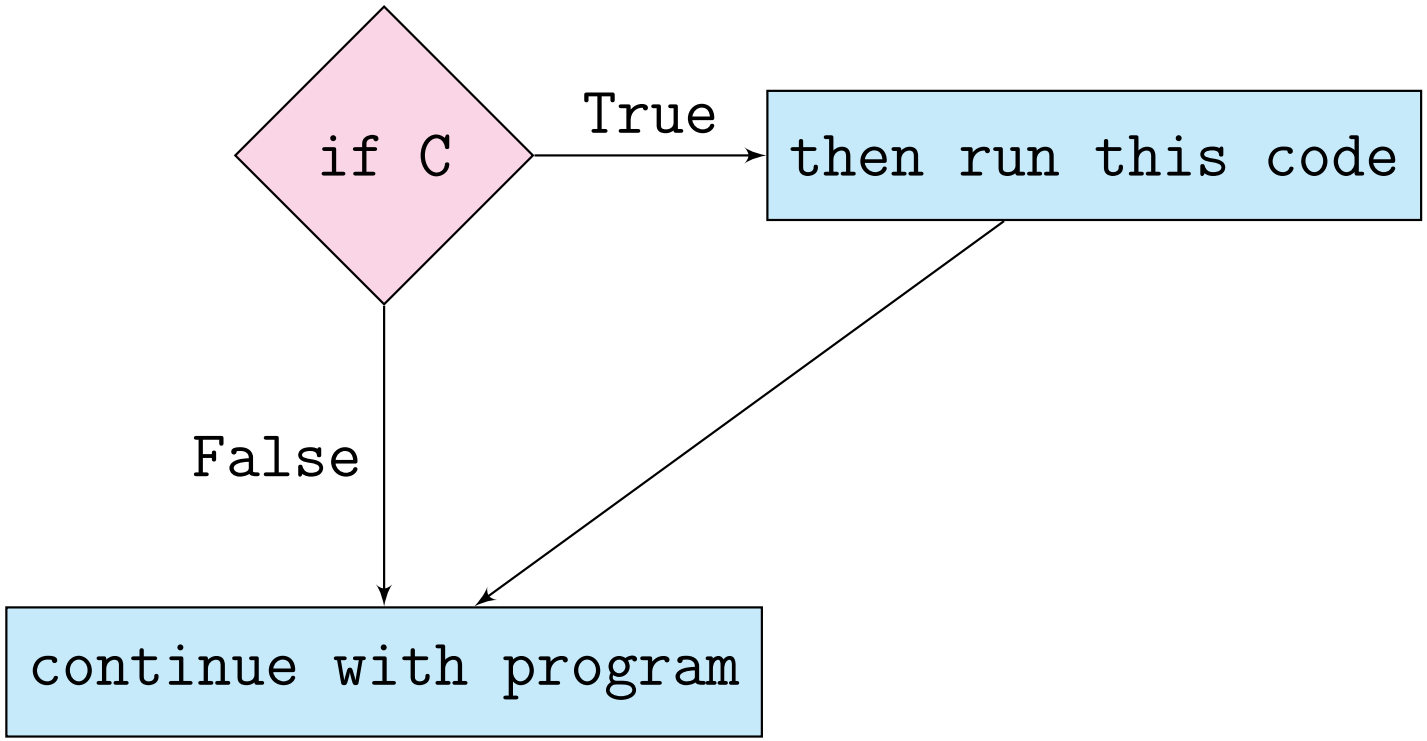
Definition 5 (If-statement)

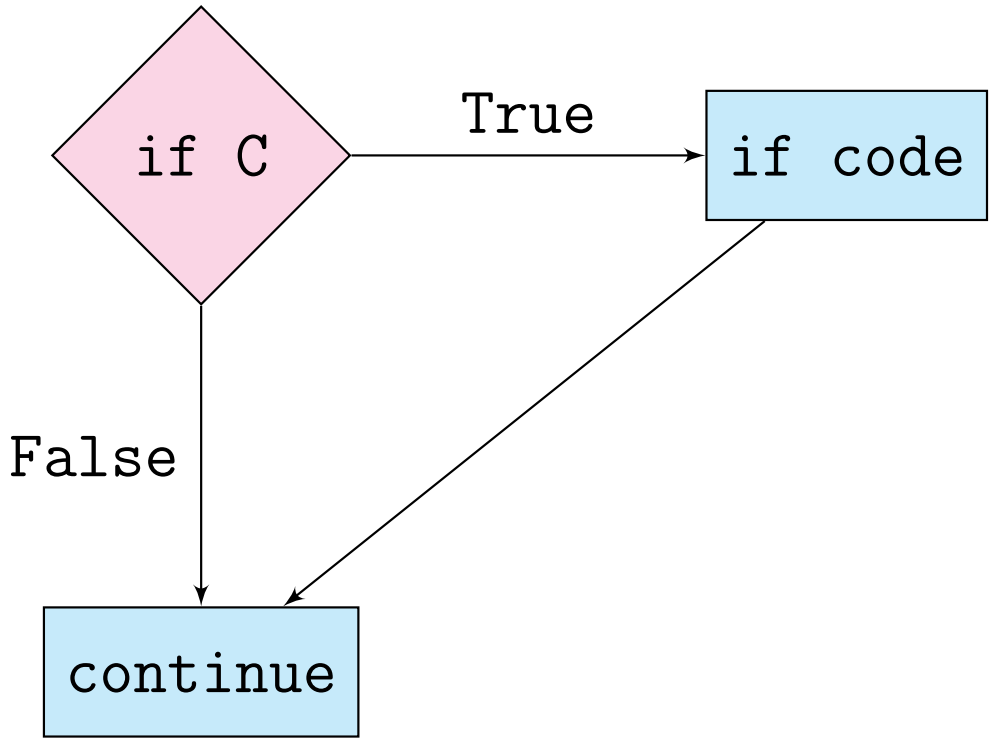
Given a condition or predicate statement C (i.e. something that evaluates to a boolean) an if-statement is a control structure that executes a block of code when C is True.

If-Then

```
1 if <cond>:
2     <code executed when cond == True>
3       ⋮
```

In Python spaces matter — only code indented within an if-statement gets executed.


if C
True
then run this code
False
continue with program

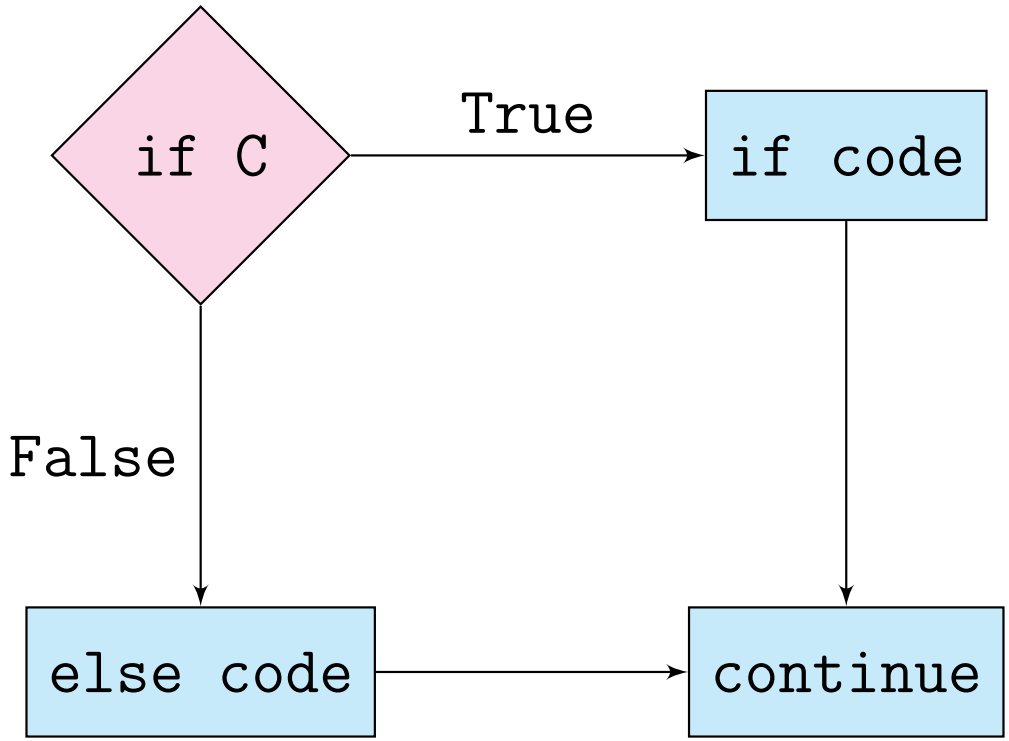

if C
True
if code
False
continue

```
>>> x = 1
>>> if 0 == 7:
...     x = x + 1
>>> x
1

>>> xs = "hello world"
>>> if 'h' in xs:
...     xs = "goodbye" + xs[-6:]
>>> xs
'goodbye world'
```

If-Then-Else

```
1 if <cond>:
2     <code>
3 else:
4     <code>
```

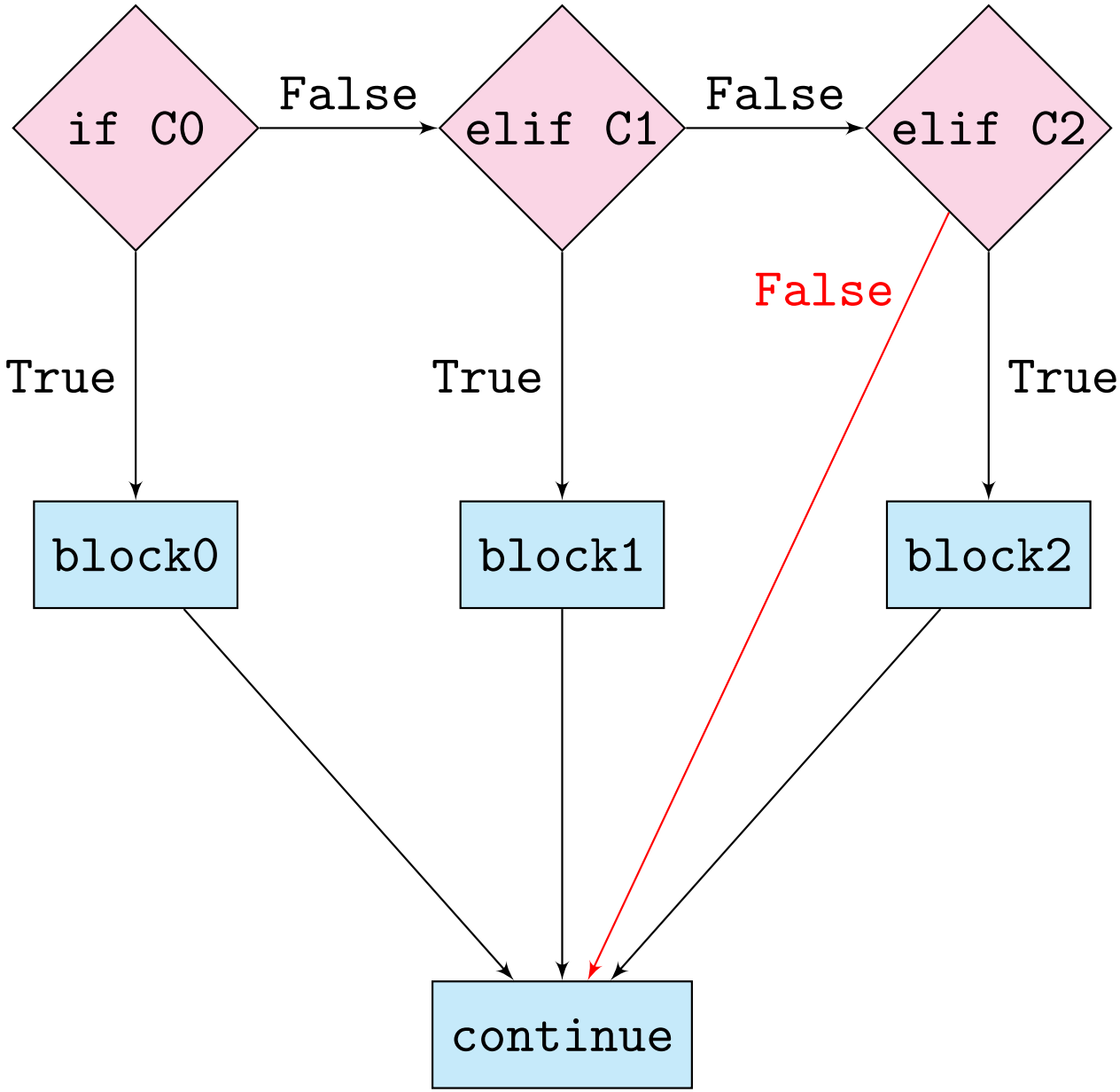

if C
True
if code
False
else code
continue

```
>>> if balance >= 0:
...     in_the_black = True
...     in_the_red = False
... else:
...     in_the_black = False
...     int_the_red = True
```

Now "extra" code executes regardless of the truth of C.

Else-If

```
1 if <cond0>:
2     <code>
3 elif <cond1>:
4     <code>
5   :
6 elif <condN>:
7     <code>
```


if C0
False
elif C1
False
elif C2
True
True
False
True
block0
block1
block2
continue

```
>>> x = True
>>> y = False

>>> if not x:
...     ans = "panda"
>>> elif x and y:
...     ans = "snake"
>>> elif not x or y:
...     ans = "badger"

>>> ans
NameError: name 'ans' is not defined
```

Definition 6 (Loop)

A loop is a control structure that repeats code that belongs to it.

Definition 7 (For-Loop)

A for-loop is a control structure that, given a group (or "collection"), repeats code for every member of that group in order.

Definition 8 (For-Loop)

```
for <name> in <iterator>:
    <code>
```

```
>>> for ch in "abcd":
...     print(ch)
a
b
c
d
>>> ch
'd'
```

Notice the order.

Definition 9 (Accumulator)

An accumulator is a variable that a loop uses to 'accumulate' an aggregate value.

```
1 >>> acc = ""
2 >>> for ch in "abcd":
3 ...     acc = acc + ch
4 ...     print(acc)
5 a
6 ab
7 abc
8 abcd
```

Accumulating

We can use loops to accumulate something as in:

```
1 >>> ans = ""
2 ... for x in "abcdef":
3 ...     ans = ans + x
```

Because this is done so much there is a short form for it:

```
1 >>> ans = ""
2 ... for x in "abcdef":
3 ...     ans += x
```

Another way of using For Loops

```
1 >>> word = "abdef"
2 >>> acc = ""
3 ... for i in range(len(word)):
4 ...     acc = acc + word[i]
5 >>> print(acc)
6 'abcdef'
```

Definition 10 (Python Assignment Operators)

1.	`x += y`	is equivalent to	`x = x + y`
2.	`x *= y`	is equivalent to	`x = x * y`
3.	`x /= y`	is equivalent to	`x = x / y`
4.	`x %= y`	is equivalent to	`x = x % y`

Definition 11 (While-Loop)

A while-loop is a control structure that repeats code while some condition is satisfied.

Definition 12 (While-Loop)

```
1 while <condition>:
2     <code>
```

```
>>> x = 0
>>> while x < 10:
...     x += 1

>>> x
10
```

```
>>> x = 0
>>> while True:
...     print(x)
...     x += 1
0
1
2
3 ...
```

There is usually a key-stroke, typically ctrl + c, *that terminates a loop — it is a good idea to learn what it is in your IDE!*

```
>>> x = 0
>>> while False:
...     print(x)
...     x += 1
```

Nothing prints.

Definition 13 (List)

A list is an ordered sequence of elements. These elements are not necessarily the same type.

Square brackets `[]` are used to create lists in Python.

```
>>> xs = [1, 2]
>>> xs[0]
1
>>> xs[0] = 2*xs[1]
>>> xs[0]
4
```

Note: Lists are mutable.

Looping Over Lists

```
1 >>> xs = ['a', 'b', 'c', 'd', 'e']
2 >>> for x in xs:
3 ...     print(x)
4 a
5 b
6 c
7 d
8 e
```

Comparison

```
>>> [1,2,3] < [4,5,6]
True

>>> [7,2,3] < [4,5,6]
False
```

Point-wise comparisons from position zero.

```
>>> [] < [1]
True
```

Membership

```
1 >>> 1 in [1, 2, 3]
2 True
3
4 >>> 0 in [1, 2, 3]
5 False
6
7 >>> [1] in [1, 2, 3]
8 False
```

List Built-Ins

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__gt__',
'__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

Append

```
>>> xs = [0, 1, 2]
>>> xs.append(3)
>>> xs
[0, 1, 2, 3]          # This is the original list updated
```

code similar to...

```
>>> xs = [0, 1, 2]
>>> xs = xs + [3]
>>> xs
[0, 1, 2, 3]
```

Extend

```
>>> xs = [1, 2, 3]

>>> xs.append([4,5])
>>> xs
[1, 2, 3, [4,5]]
```

Extend

```
1 >>> xs = [1, 2, 3]
2
3 >>> xs.extend([4,5])
4 >>> xs
5 [1, 2, 3, 4, 5]
6
7  code similar to...
8 >>> xs =  xs + [4, 5]
```

Copying Lists

Take care when copying lists.

```
1 >>> xs = [1, 2, 3]
2
3 >>> ys = xs
4 >>> ys[-1] = 9
5
6 >>> xs
7 [1, 2, 9]
```

Copying Lists

```
>>> xs = [1, 2, 3]

>>> ys = xs.copy()
>>> ys[-1] = 9

>>> xs
[1, 2, 3]

>>> ys
[1, 2, 9]
```

Slicing

```
1  >>> xs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2
3  >>> xs[3:6]
4  [3, 4, 5]
5
6  >>> xs[::2]
7  [0, 2, 4, 6, 8]
8
9  >>> xs[7:2:-2]
10 [7, 5, 3]
```

Definition 14 (Range)

Python's `range` keyword allows us to quickly build an iterator for use by for-loops.

It has general form:

```
range([start], stop[, step])
```

which is similar to list slicing.

```
>>> range(10)
range(0, 10)

>>> type(range(10))
<class 'range'>

>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> list(range(2, 7))
[2, 3, 4, 5, 6]

>>> list(range(2, 7, 3))
[2, 5]

>>> list(range(2, 7, -1))
[]

>>> list(range(7, 2, -1))
[7, 6, 5, 4, 3]
```

`range` can be used to walk through lists using an index.

```
1 >>> xs = [1, 2, 3, 4]
2 >>> for k in range(len(xs)):
3 ...     #Code involving xs[k]
```

Nested Lists

A list can have another list as an element

```
1 >>> xs = [ [1,2], [5,6,7] ]
2 >>> len(xs)
3 2
4 >>> len( xs[-1] )
5 3
```

Nested Loops

Recall that loops can be nested.

```
1 >>> X = [ [1, 2], ["a", "b", "c"], "hello" ]
2 >>> for xs in X:
3 ...     for x in xs:
4 ...         print(x, end=' ')
5 1 2 a b c h e l l o >>>
```

Definition 15 (Dictionary)

A dictionary is an unordered collection of objects. Curly (or squiggly) brackets {} are used to create dictionaries in Python, which are accessible via a "key" mapping (`:`) to a "value", separated by a comma (`,`).

Dictionary

```
>>> d = "apples":12, "pears":20, "pineapples":7
>>> d
‘apples’: 12, ‘pears’: 20, ‘pineapples’: 7
>>> d["apples"]
12
>>> d["apples"] = 30
>>> d
‘apples’: 30, ‘pears’: 20, ‘pineapples’: 7
```

Dictionary Built-Ins

```
>>> dir(dict)
['__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__getitem__', '__gt__', '__hash__',
 '__init__', '__init_subclass__', '__iter__', '__le__',
 '__len__', '__lt__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'clear',
 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop',
 'popitem', 'setdefault', 'update', 'values']
 >>> help(dict)
```

Random Number Generation

Generating truly random numbers from software alone is impossible (it is possible with hardware).

Software generates pseudorandom numbers meaning they appear random but are generated by deterministic methods.

Consequently, you can generate the same sequence of random numbers (useful for testing) using `random.seed`.

Random number generation is handled by an external library. It is not built-in and therefore must be imported.

Single Command

```
1 >>> from random import randint
2 >>> randint(1,6)          Chosen uniformly over the interval.
3 2
```

Entire library

```
1 >>> import random
2 >>> random.randint(1,6)
3 2
```

```
>>> random.seed(1)
>>> random.randint(1, 10**4)
2202
>>> random.randint(1, 10**4)
9326

>>> random.seed(1)
>>> random.randint(1, 10**4)
2202
>>> random.randint(1, 10**4)
9326
```

Definition 16 (CSV Files)

`Comma Separated Value (CSV) Files` are a type of plain text file that uses specific structuring to arrange tabular data.

The structure looks something like this:

```
column 1 name, column 2 name, column 3 name
first row data 1, first row data 2, first row data 3
second row data 1, second row data 2, second row data 3
...
nth row data 1, nth row data 2, nth row data 3
```

CSV Files: General & Reading

CSV manipulation is handled by an external library. It is not built-in and therefore must be imported.

```
1 import csv
2
3 with open("people.csv") as csv_file:
4   csv_reader = csv.reader(csv_file, delimiter=",")
5   for row in csv_reader:
6     print(row)
7
8   csv_file.close()
```

CSV Files: Storing Data

```
1 import csv
2 data = []
3 with open("people.csv") as csv_file:
4   csv_reader = csv.reader(csv_file, delimiter=",")
5   line_count = 0
6   for row in csv_reader:
7     data.append(row)
8   csv_file.close()
```

CSV Files: Writing

```
1 import csv
2 with open("output.csv", mode="w") as out_file:
3   csv_writer = csv.writer(out_file, delimiter=",")
4   csv_writer.writerow(data[0]) # write header row
5   for row in data:
6     entry = []
7     entry.append(row[n])
8     csv_writer.writerow(entry)
9   out_file.close()
```

Analysis of Data

Once data has been read in, regardless of the mechanism, you can perform analysis of that information!

For example:

1. Regression
2. Clustering
3. pandas' Python Data Analysis Library
4. sklearn's Machine Learning Library

Graphing in Python

There are numerous applications and methods in which you can graph data. In Python, you can use:

- matplotlib – ***this will be demonstrated today!***
- pandas – built-off of matplotlib
- seaborn
- Plotly Python

Matplotlib: Scatterplot

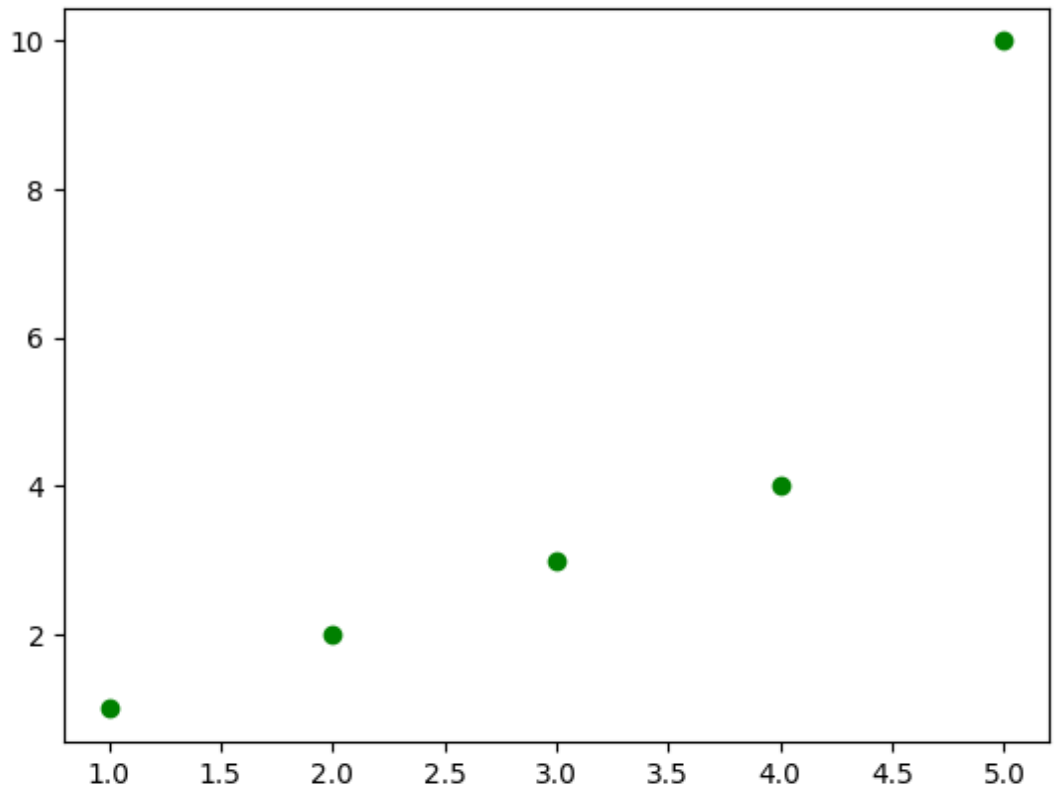
```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4,5], [1,2,3,4,10], 'go')
3 plt.show()
```

Note: if you were to remove the "go" you'd have a line chart.

Interesting enough, the format go- is actually a 3-character code representing green coloured dots with a solid-line.

For a complete list of colors, markers and linestyles, check out the help(plt.plot)

Matplotlib: Scatterplot

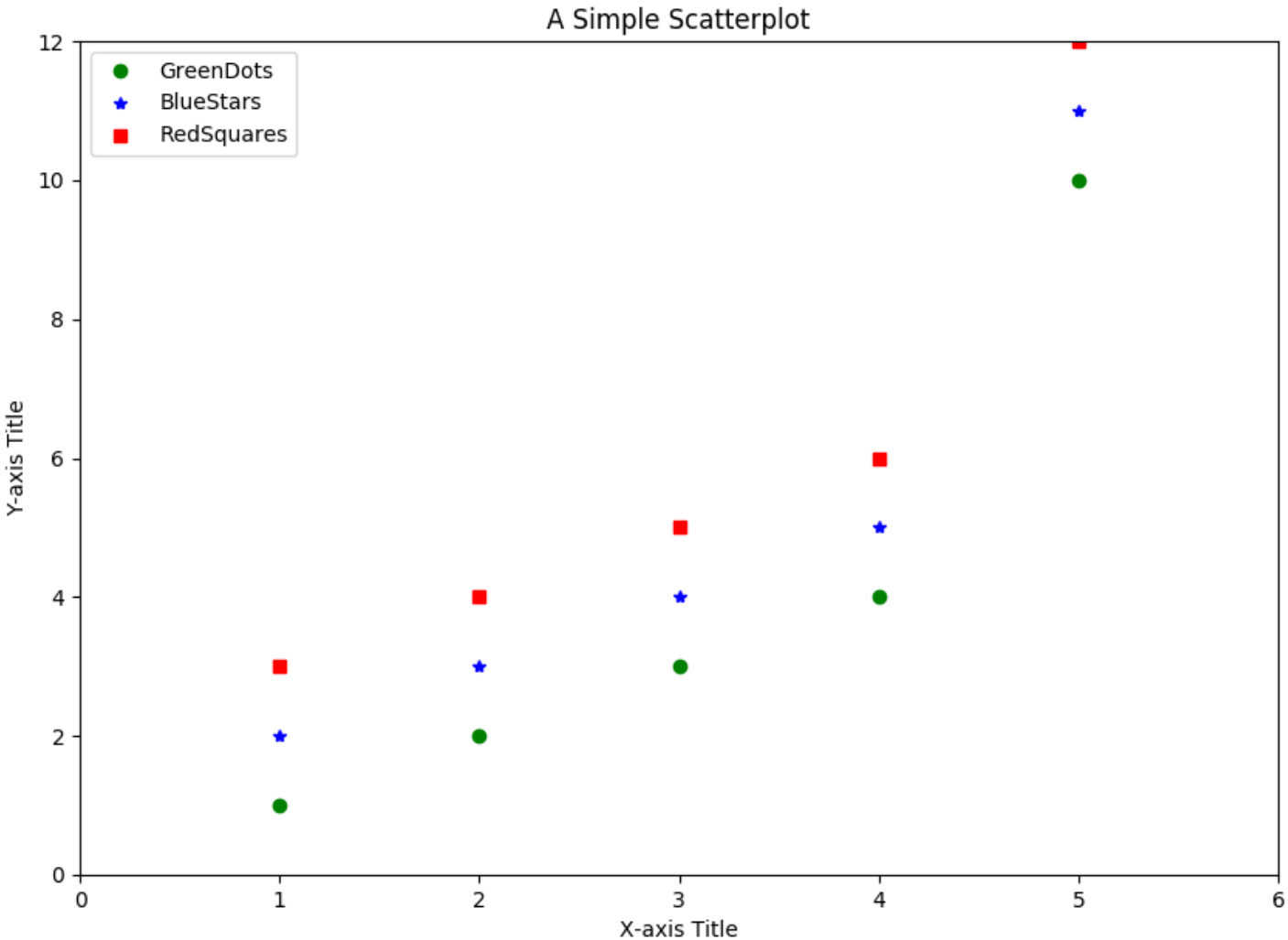


Notice the lack of title, axis names, and legend.

Matplotlib: Scatterplot

```
1 fig = plt.figure(figsize=(10,7))  # 10=width, 7=height
2 fig.canvas.set_window_title("My Figure Name")
3 plt.plot([1,2,3,4,5],[1,2,3,4,10],‘go’,label=‘GreenDots’)
4 plt.plot([1,2,3,4,5],[2,3,4,5,11],‘b*’,label=‘BlueStars’)
5 plt.plot([1,2,3,4,5],[3,4,5,6,12],‘rs’,label=‘RedSquares’)
6 plt.title(‘A Simple Scatterplot’)
7 plt.xlabel(‘X-axis Title’)
8 plt.ylabel(‘Y-axis Title’)
9 plt.legend(loc=‘best’)
10 plt.show()
```

Matplotlib: Scatterplot



Thank You!

Thanks for listening! I'm always happy to answer questions! :)

Feel free to email me: michael.liut@utoronto.ca