

Python Fundamentals ODSC Europe 2023

Philip Tracton

Lecture Summary

- Intro and Background
- Tools and Installation
- Basic Python

Introduction

- Instructor: Philip Tracton
- BS in EE from University of Maryland
- MS in EE from California State University at Northridge
- Works on embedded software, ASIC, FPGA and DSP
- Employed at Medtronic for over 20 years.
- Teaching at UCLA Extension since 2010.

Course Goals

- 1 Learn Python basics
- 2 Become familiar with Jupyter Labs
- 3 Be able to expand your learning on your own from here

Tools We Will Need

- 1 Python Interpreter and Libraries
- 2 Jupyter Labs

- We will be using the latest version of Anaconda Python
 - Free
 - Open Source
 - Runs on Windows, Mac and Linux
 - Comes with many popular 3rd party libraries
 - Comes with conda – a tool to manage the libraries

Jupyter Lab is an interactive environment for running python code. It is very common in the data science world for exploring code and data.

White Space

- PEP 8 rules on White Space
- Most controversial aspect of Python.
- White space used to distinguish blocks
- Tab == Space!
- Comments and empty lines are ignored
- Avoid using Tab
- Use an editor/IDE that is Python aware

Code Example:

```
3
4
5
6
7
↪
8
9
10
11
12
13
14
15
16
17
18
```

```
def factorial(n=1):
    """
    This is the function's comment, it explains
    what the function does, how it does it and why
    it
    does it
    """
    if n == 1:
        return n
    else:
        return n * factorial(n - 1)

if __name__ == "__main__":
    # Single line comment example!
    print(factorial(5))
```


Coding Standards

- Python has specified the coding standards for code written in this language
- This is PEP 8
- If you are in doubt about a style issue with the way you write Python, refer to PEP 8 and follow it

Comments

- Python comments start with `'#'`
- Multi-line comments can be put in triple quotes `""" THIS IS A COMMENT """`

Duck Typing

- “If it walks like a duck and quacks like a duck...”
- No type checking unless absolutely necessary
- Python will let you change the data type of a variable at any time!

Duck Typing

- “If it walks like a duck and quacks like a duck...”
- No type checking unless absolutely necessary
- Python will let you change the data type of a variable at any time!

WARNING

BE CAREFUL!

Variables can change type if you are not careful

Variables

- PEP 8 Rules on variable names
- Create variable with assignment:
 - `a = 5`
- Variables are references!
- Variables are memory managed
- Data pointed to by variable is destroyed when variable goes out of scope
- Assigning a variable to another creates new reference
 - `b = a`

Integer Data Types

Official documentation

- Integers have unlimited precision.
- Many built in functions
- Tools to convert formats

Integer Operations

$x+y$	Addition of x and y
$x-y$	Subtraction of y from x
$x * y$	Multiplication of x and y
x/y	Divides x by y and returns a float
$x//y$	Divides x by y and truncates fractional part to return an integer
$x\%y$	Returns the modulus (remainder) of x divided by y
$x**y$	Raises x to the power of y
$-x$	negates x

Integer Functions

<code>abs(x)</code>	Returns the absolute value of <code>x</code>
<code>divmod(x,y)</code>	Returns the quotient and remainder of <code>x</code> divided by <code>y</code> as a tuple
<code>pow(x, y)</code>	Raises <code>x</code> to the power of <code>y</code> , same as <code>x ** y</code>
<code>pow(x,y,z)</code>	Faster alternative to <code>(x ** y)%z</code>
<code>round(x, n)</code>	Returns <code>x</code> rounded to <code>n</code> integral digits
<code>bin(x)</code>	Returns a string that is a binary representation of <code>x</code> , <code>bin(5) == "0b101"</code>
<code>hex(x)</code>	Returns a string that is a hexadecimal representation of <code>x</code> , <code>hex(30) == "0x1e"</code>
<code>int(x)</code>	Converts object <code>x</code> to an integer or raises an error
<code>int(s, base)</code>	converts string <code>s</code> to an integer
<code>oct(x)</code>	Returns a string that is a octal representation of <code>x</code> , <code>oct(30) == "036"</code>

Floating Point

- Holds double precision values
- Range depends on compiler and platform that we are running
- Float performs all the same operations as the integer type
- `float_var = 1.0`

This is Python 3 Strings Documentation

- immutable sequence of Unicode characters
- Arbitrary length
- Support usual comparison operations
- Can use single, double or tripple quotation marks to create a string
- `string_var = 'This is a string'`
- `string_var = "This is a string"`
- `string_var = '''This is a string'''`

String Operations

<code>s[x]</code>	Access index <code>x</code> in the string
<code>s[:x]</code>	Access from the start of the string to index <code>x</code>
<code>s[x:]</code>	Access from index <code>x</code> to the end of the string
<code>s[x:y]</code>	Access from index <code>x</code> to index <code>y</code>
<code>s[x:y:z]</code>	Access from index <code>x</code> to index <code>y</code> by steps of <code>z</code>
<code>s1+s2</code>	Concatenation of 2 strings
<code>s * n</code>	Multiply the string <code>n</code> times, creates a new string with <code>n</code> copies of the old string
<code>len(s)</code>	Length of string <code>s</code>
<code>c in s</code>	Returns a Boolean <code>True</code> if <code>c</code> is in <code>s</code>
<code>c not in s</code>	Returns a Boolean <code>True</code> if <code>c</code> is not in <code>s</code>

String Methods

This is a list of the commonly used methods.
The full list [starts here](#).

<code>str.capitalize()</code>	Capitalize the first character
<code>str.format(*args, **kwargs)</code>	Reformat the string with the specified parameters
<code>str.join(iterable)</code>	Concatenate a list of strings
<code>str.lower()</code>	Returns strings as all lower-case characters
<code>str.split(sep=None, maxsplit=-1)</code>	split the string into a list of strings based on sep delimiter
<code>str.strip([chars])</code>	Return the string with those characters removed.

- Official Documentation
- List contains arbitrary objects
- Mutable
- Can be sorted
- `list_var = []` # Creates an empty list
- `list_var = [1,2,"String"]` # Creates an list with data

List Operations

<code>s[x]</code>	Access index x in the list
<code>s[:x]</code>	Access from the start of the list to index x
<code>s[x:]</code>	Access from index x to the end of the list
<code>s[x:y]</code>	Access from index x to index y
<code>s[x:y:z]</code>	Access from index x to index y by steps of z
<code>s1+s2</code>	Concatenation of 2 lists
<code>s * n</code>	Multiply the list n times, creates a new list with n copies of the old list
<code>len(s)</code>	Length of list s
<code>c in s</code>	Returns a Boolean True if c is in s
<code>c not in s</code>	Returns a Boolean True if c is not in s

List Methods

This is a list of the commonly used methods.
The full list [starts here](#).

<code>list.append(x)</code>	add x onto the end of the list
<code>list.extend(L)</code>	extend the list with a new list L
<code>list.insert(i,x)</code>	insert x at position i in the list
<code>list.pop(i)</code>	remove the item from position i, if i is not specified, remove the last item
<code>list.sort()</code>	sort items in place

Dictionaries

- Official Documentation
- Dictionaries contain arbitrary key/value pairs
- Like an associative array
- Keys must be unique and hashable
- Lookup is $O(1)$
- No sorting order
- Dictionary is mutable!
- Literals:
 - `dict_var = {key1:value1, key2:value2...}`
 - `empty_dict = { }`

Dictionary Operations and Methods

<code>len(d)</code>	Returns the number of key/value pairs
<code>value = d[key]</code>	Returns the value for this key, an error if key does not exist
<code>d[key] = value</code>	Creates or replaces this key and associates this data with it
<code>del d[key]</code>	Deletes this key/value
<code>key in d</code>	Returns True if it is, False if it is not
<code>d.keys()</code>	Returns a list of keys
<code>d.values</code>	Returns a list of values
<code>d.update(dict)</code>	Adds the passed in dictionary to d
<code>d.items()</code>	Returns a list of key/value tuples

LAB 1

Lecture Summary

- If-Else
- For Loops
- While Loops
- Functions

If Statements

- Official Documentation
- elif and else are optional

Code Example:

```
3  if __name__ == "__main__":  
4      x = 4  
5      if x == 4:  
6          print("We are 4")  
7      elif x == 5:  
8          print("We are 5")  
9      else:  
10         print("Default to something else")
```

For Statements

- Official Documentation
- Do not specify start, stop and step like C
- Iterate over a list, file or string
- range function returns a list of numbers to iterate over
- break will exit the loop
- else: will execute at the end of the loop if the exit is normal (not break)

Code Example:

```
3  if __name__ == "__main__":
4      languages = ["python", "java", "c", "lisp",
5                  ↪ "perl"]
6      for x in languages:
7          print(x)
8      for x in range(10):
9          print(x)
10     else:
11         print("Done")
12     for x in range(10):
13         print(x)
14         if x > 4:
15             break
16     else:
17         print("Done")
18 my_dict = {1: "One", 2: "Two", 3: "Three"}
19 for k, v in my_dict.items():
    print(k, v)
```

While Statements

Code Example:

- Official Documentation
- As long as the expression in the while statement evaluates to True, the loop will process
- break can exit the loop
- else: will execute at the end of the loop if the exit is normal (not break)

```
3  if __name__ == "__main__":
4      languages = ["python", "java", "c", "lisp",
5                  ↪ "perl"]
6      count = 0
7      while count < len(languages):
8          print(languages[count])
9          count += 1
10     count = 0
11     while count in range(10):
12         print(count)
13         count += 1
14     else:
15         print("Done")
16     count = 0
17     while count in range(10):
18         print(count)
19         count += 1
20         if count > 4:
21             break
22     else:
23         print("Done")
```

Basic Exceptions

- Exceptions are python's way of handling errors
- Exception Documentation
- Exception Tutorial
- You must have try and except, all other elements are optional
- Each exception block is tried in order
- Use liberally and do not let errors go silently
- A blank except: will catch any exception.

Basic Exceptions

- Exceptions are python's way of handling errors
- Exception Documentation
- Exception Tutorial
- You must have try and except, all other elements are optional
- Each exception block is tried in order
- Use liberally and do not let errors go silently
- A blank except: will catch any exception.

```
3
4 def key_trigger(key=None):
5     d = {1: "One"}
6     return d[key]
7
8
9 def trigger(x, y):
10     return x / y
11
12
13 if __name__ == "__main__":
14
15     try:
16         print(key_trigger(1))
17         print(key_trigger()) # KeyError
18         print(trigger(1, 2))
19         print(trigger(1, 0)) # ZeroDivisionError
20     except ZeroDivisionError as e:
21         print("Exception: %s" % (e.args))
22     except KeyError:
23         print("Key Error")
24     else:
25         print("All is well")
26     finally:
27         print("Finally")
```


Functions

- Functions are a way of breaking up the code into more manageable pieces.
- There are 4 types of Functions

Functions

- Functions are a way of breaking up the code into more manageable pieces.
- There are 4 types of Functions
 - 1 Global are available to everyone in the module

Functions

- Functions are a way of breaking up the code into more manageable pieces.
- There are 4 types of Functions
 - 1 Global are available to everyone in the module
 - 2 Local are functions inside of functions

Functions

- Functions are a way of breaking up the code into more manageable pieces.
- There are 4 types of Functions
 - 1 Global are available to everyone in the module
 - 2 Local are functions inside of functions
 - 3 Lambda are limited functions that are created just in time to use them

Functions

- Functions are a way of breaking up the code into more manageable pieces.
- There are 4 types of Functions
 - 1 Global are available to everyone in the module
 - 2 Local are functions inside of functions
 - 3 Lambda are limited functions that are created just in time to use them
 - 4 Methods are associated with a specific data type and part of Object Oriented Programming

Functions

- Functions are a way of breaking up the code into more manageable pieces.
- There are 4 types of Functions
 - 1 Global are available to everyone in the module
 - 2 Local are functions inside of functions
 - 3 Lambda are limited functions that are created just in time to use them
 - 4 Methods are associated with a specific data type and part of Object Oriented Programming

Pseudo Code Example:

```
3 def function_name ( < optional list of parameters  
↳ > ):  
4     """  
5     Function doc string that explains how the  
↳ function  
6     works  
7     """  
8     ... code goes here...  
9     return < optional return value >
```

Names and DocStrings

- PEP 8 rules on Function Names
- Use good clear names for the functions, should indicate what the function does
- Avoid abbreviations
- Docstrings are comments that come right after the def line of the function. It should indicate what the function does and how to use it.
- Docstrings have their own PEP, PEP 257
- Although PEP 8 also has some thoughts

Names and DocStrings

- PEP 8 rules on Function Names
- Use good clear names for the functions, should indicate what the function does
- Avoid abbreviations
- Docstrings are comments that come right after the def line of the function. It should indicate what the function does and how to use it.
- Docstrings have their own PEP, PEP 257
- Although PEP 8 also has some thoughts

All functions in your labs must have doc strings!

Function Arguments

- Does not type check
 - you can manually check if you wish
- All values are passed by reference
 - immutable types can't be modified by a function
- Arguments are local variables
- Arguments go out of scope when functions return
- Without a return statement, the function returns **None**

Default Arguments

- Specify a default value for a function argument
- If no value is given for this parameter when the function is called, the default value is used
- This is very handy!

Default Arguments

Code Example:

- Specify a default value for a function argument
- If no value is given for this parameter when the function is called, the default value is used
- This is very handy!

```
3      """
4      This Is An Example Program For Learning Python
5      """
6
7
8      def function_example(param1="Hi", param2=False):
9          """
10             This is the function's doc string,
11             it is a comment that explains the
12             function and how to use it.
13
14             This function takes 2 parameters (param 1 and
15             ↪ param2)
16             and prints them out.
17
18             EX: function_example("1", "One")
19             """
20             print(param1, param2)
21             return
22
23     if __name__ == "__main__":
```

Named Arguments

- Use name arguments to comment the code (use good names!)
- Can be used to skip arguments

Named Arguments

Code Example:

- Use name arguments to comment the code (use good names!)
- Can be used to skip arguments

```
3      """
4      This Is An Example Program For Learning Python
5      """
6
7
8      def function_example(param1="Hi", param2=False):
9          """
10             This is the function's doc string,
11             it is a comment that explains the
12             function and how to use it.
13
14             This function takes 2 parameters (param 1 and
15             ↪ param2)
16             and prints them out.
17
18             EX: function_example("1", "One")
19             """
20             print(param1, param2)
21             return
22
23     if __name__ == "__main__":
24         function_example()
25         function_example("1", "One")
26         function_example(param2="there!")
27         function_example(param2="order",
28             ↪ param1="Reverse")
```

LAB 2

Lecture Summary

- import
- numpy
- pandas
- Matplotlib

- The Python Import System
- This is how we bring in code from one module to another
- Gain access to built in python libraries
- Gain access to downloaded 3rd party libraries
- Create your own to organize your code

LAB 3