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 Intepreter and Libraries
2. Jupyter Labs

Python Interpreter

- 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

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 and editor/IDE that is Python aware

Code Example:

```python

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

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

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

Strings

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

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

String Methods

This is a list of the commonly used methods.
The full list starts here.

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

Lists

- 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

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

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

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

Lab

LAB 1

Lecture Summary

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

If Statements

- Official Documentation
- elif and else are optional

Code Example:

```python
if __name__ == "__main__":
    x = 4
    if x == 4:
        print("We are 4")
    elif x == 5:
        print("We are 5")
    else:
        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:

```
if __name__ == "__main__":
    languages = ["python", "java", "c", "lisp",
    ↪   "perl"]
    for x in languages:
        print(x)
    for x in range(10):
        print(x)
    else:
        print("Done")
    for x in range(10):
        print(x)
        if x > 4:
            break
    else:
        print("Done")
    my_dict = {1: "One", 2: "Two", 3: "Three"}
    for k, v in my_dict.items():
        print(k, v)
```

While Statements

- 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)

Code Example:

```python
if __name__ == "__main__":
    languages = ["python", "java", "c", "lisp",
    ↪  "perl"]
    count = 0
    while count < len(languages):
        print(languages[count])
        count += 1
    count = 0
    while count in range(10):
        print(count)
        count += 1
    else:
        print("Done")
    count = 0
    while count in range(10):
        print(count)
        count += 1
        if count > 4:
            break
    else:
        print("Done")
```

Basic Exceptions

- Exceptions are pythons 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 pythons 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.

```python

def key_trigger(key=None):
    d = {1: "One"}
    return d[key]


def trigger(x, y):
    return x / y


if __name__ == "__main__":

    try:
        print(key_trigger(1))
        print(key_trigger())  # KeyError
        print(trigger(1, 2))
        print(trigger(1, 0))  # ZeroDivisionError
    except ZeroDivisionError as e:
        print("Exception: %s" % (e.args))
    except KeyError:
        print("Key Error")
    else:
        print("All is well")
    finally:
        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

Psuedo Code Example:

```
def function_name ( < optional list of parameters > ):
    """
    Function doc string that explains how the function
    works
    """
    ... code goes here...
    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 returrns **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

- 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!

Code Example:

```python
"""
This Is An Example Program For Learning Python
"""


def function_example(param1="Hi", param2=False):
    """
    This is the function's doc string,
    it is a comment that explains the
    function and how to use it.

    This function takes 2 parameters (param 1 and
↪   param2)
    and prints them out.

    EX: function_example("1", "One")
    """
    print(param1, param2)
    return


if __name__ == "__main__":
```

Named Arguments

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

Named Arguments

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

Code Example:

```python
"""
This Is An Example Program For Learning Python
"""


def function_example(param1="Hi", param2=False):
    """
    This is the function's doc string,
    it is a comment that explains the
    function and how to use it.

    This function takes 2 parameters (param 1 and
↪   param2)
    and prints them out.

    EX: function_example("1", "One")
    """
    print(param1, param2)
    return


if __name__ == "__main__":
    function_example()
    function_example("1", "One")
    function_example(param2="there!")
    function_example(param2="order",
    ↪   param1="Reverse")
```

Lab

LAB 2

Lecture Summary

- import
- numpy
- pandas
- Matplotlib

Lab

LAB 3