PYTHON FOR DATA SCIENCE AND MACHINE LEARNING

BY ZEPHANIA REUBEN

Outline:

- Basics
- Collection Data Types
- Functions
- Object Oriented Python

Get started

What is Python?

- High-level,
- General purpose,
- Interpreted,
- Interactive and object-oriented scripting language.
- Python is designed to be highly readable

Features of Python

- Easy to learn
- Easy to read
- Easy to maintain
- A broad standard libraries
- Interactive mode

Why Python?

"Why Python?“

- The answer is simple: it is powerful yet very accessible.
- Also Python has become the most popular programming language for data science because it allows us to forget about the tedious parts of programming and offers us an environment where we can quickly jot down our ideas and put concepts directly into action.

Python installation

- To install Python open a web browser go to https://www.python.org/downloads.
- Run the downloaded file.
- Just accept the default settings and wait until the install is finished.

Python IDLE

- **IDLE** is a simple integrated development environment (IDE) that comes with Python.
- It’s a program that allows us to type in our programs and run them.
- You can find IDLE in the Python 3.7 folder on your computer.

Cont…

- When is started , the IDLE starts up in the shell, which is an interactive window where we can type in Python code and see the output in the same window.
- Most of the time we will want to open up a new window and type the program in there.

Jupyter Notebook

- We can use another Python platform called Anaconda.

- It includes different of popular data science packages and virtual environment manager..

Cont..

- **Jupyter Notebook** Is an open source web application interactive and exploratory computing.
- It allows us to create and share documents that contain live code, equations, visualizations and explanatory text.
- We will work in **Jupyter notebooks** for all practices

Running Python

Python program can be executed in different modes such as: -

- Interactive mode
- Script mode

Python identifiers

- Variable names can contain letters, numbers, and the underscore.
- Variable names cannot contain spaces
- Variable names cannot start with a number.
- Case matters—for instance, temp and Temp are different.

Python keywords

and	is	not	exec	lambda	for	def	if	return	def
import	try	elif	in	print	raise	while	else	with	except
yield	assert	finally	or	break	pass	class	from	continue	global

Lines and Indentation

- Python provides no braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation.
- The number of spaces in the indentation is variable.
- All statements within the block must be indented the same amount.

Quotation in Python

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines.

Comments in Python

- A hash sign (#) that is not inside a string literal begins a comment.
- All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Printing

- The **print** function requires parenthesis around its arguments.
- To print several things at once, separate them by **commas**. Python will automatically insert spaces between them.
- Python will insert a space between each of the arguments of the print function.

Cont…

- There is an optional argument called **sep**, short for **separator**, that we can use to change that space to something else.
- For example, using **sep='##'** would separate the arguments by two **pound signs**.
- The print function will automatically advance to the next line.

Variable Types

- Variables are nothing but reserved memory locations to store values
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

Standard data types in Python

Python has five standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary

Python numbers

Number data types store numeric values. Number objects are created when you assign a value to them.

Python supports four different numerical types:-

- **int** (signed integers)
- **long** (long integers, they can also be represented in octal and hexadecimal)
- **float** (floating point real values)
- **complex** (complex numbers)

Python strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:])
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Python Tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas.
- The main differences between lists and tuples are:
- Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as **read-only** lists.

Python Dictionary

- Python's dictionaries consist of key-value pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

Data Type Conversion

- To convert between types, we simply use the type name as a function.
- There are several built-in functions to perform conversion from one data type to another.
- Example: int(), str().

Python Basic Operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Assume variable **a** holds **10** and **b** holds **20**, then:

Operator	Example
+ Addition	a+b=30
- Subtraction	a-b=10
* Multiplication	a*b=200
/ Division	a/b=2
% Modulus	b%a=0
** Exponent	a**b=10 to the power of 20

Python Comparison Operator

- These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.
- Assume variable **a** holds **10** and variable **b** holds **20**, then:

Cont…

Operator	Example
==	(a==b) is not true
!=	(a!=b) is true
<>	(a<>b) is true. This is similar to !=
>	(a>b) is not true
<	(a<b) is true
>=	(a>=b) is not true
<=	(a<=b) is true

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20,then:-

Operator	Example
=	c=a+b assigns value of a + b into c
+= Add AND	c+= a is equivalent to c=c+a
-= Subtract AND	c-= a is equivalent to c=c-a
*= Multiply	c*= a is equivalent to c=c*a
/= Divide AND	c/= a is equivalent to c=c/a
%= Modulus AND	c%= a is equivalent to c=c%a
**= Exponent AND	c**= a is equivalent to c=c**a

Python Logical Operators

- There are following logical operators supported by Python language. Assume variable **a** holds **10** and variable **b** holds **20** then:

Operator	Example
and, logical AND	(a and b) is true
or, logical OR	(a or b) is true
not , logical NOT	Not (a and b) is false

Python Membership Operators

- Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below:

Operator	Example
in	x in y , here , in results in a 1 if x is a member of sequence y
not in	x in not y , here , not results in a 1 if x is not a member of sequence y

Python Identity Operators

- Identity operators compare the memory locations of two objects. There are two Identity operators as explained below:

Operator	Example
is	x is y, here is results in 1 if id(x) equals id(y).
is not	x is not y, here is not results in 1 if id(x) is not equal to id(y).

DECISION MAKING

- Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.
- Decision structures evaluate multiple expressions which produce **TRUE** or **FALSE** as outcome.
- Python programming language assumes any **non-zero** and **non-null** values as **TRUE**, and if it is either **zero** or **null**, then it is assumed as **FALSE** value.

Cont…

- Python programming language provides following types of decision making statements.

Statement	Description
if statement	if statement consists of a Boolean expression followed by one or more statements.
if…**else** statement	if statement can be followed by an optional else statement, which executes when the Boolean expression is FALSE
Nested **if** statement	You can use one if or else if statement inside another if or else if statement(s)

LOOPS

- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- There may be a situation when you need to execute a block of code several number of times.
- A loop statement allows us to execute a statement or group of statements multiple times.

Cont…

- Python programming language provides following types of loops to handle looping requirements.

Loop type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loop	You can use one or more loop inside any another while , for or do…while loop

LOOP CONTROL STATEMENT

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Python supports the following control statements.

Cont…

Control Statement	Description
continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating
break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
pass	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

COLLECTION DATA TYPES
Lists

- The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets.
- Important thing about a list is that items in a list need not be of the same type.
- Creating a list is as simple as putting different comma-separated values between square brackets.

Accessing Values in Lists

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]

print("list1[0]: ", list1[0])
print("list2[1:5]: ", list2[1:5])
```

Basic List Operation

- Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Python Expression	Results	Description
Len([1,2,3])	3	Length
[1,2,3]+[4,5,6]	[1,2,3,4,5,6]	Concatenation
[“A”]*4	[‘A’,’A’,’A’,’A’]	Repetition
3 in [1,2,3]	True	Membership

Indexing and Slicing

- Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.
- Assume the following input:

L=[‘CIVE’, ’Cive’, ‘CIVE’]

Python	Results	Description
L[2]	‘CIVE’	Offsets start at zero
L[-2]	‘Cive’	Negative count from the right
L[1:]	[‘Cive’ , ‘CIVE’]	Slicing fetches sectors

Built-in List Functions and Methods

Python includes the following list functions:

Function	Description
cmp(list1,list2)	Compares elements of both lists.
len(list)	Gives the total length of the list
max(list)	Returns item from the list with max value.
min(list)	Returns item from the list with min value.
list(seq)	Converts a tuple into list

Cont…

Python includes the following list methods:

Methods	Description
list.append(obj)	Appends object obj to list
list.count(obj)	Returns count of how many times obj occurs in list
list.extend(seq)	Appends the contents of seq to list
list.index(obj)	Returns the lowest index in list that obj appears
list.insert(index,obj)	Inserts object obj into list at offset index
list.pop(obj=list[-1])	Removes and returns last object or obj from list
list.remove(obj)	Removes object obj from list
List.reverse()	Reverses objects of list in place

Tuples

- A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use **parentheses**, whereas lists use **square brackets**.

Accessing Values In Tiples

- To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Example:

```
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print("tup1[0]: ",tup1[0])

print("tup2[1:5]: ", tup2[1:5])
```

Dictionary

- Dictionary this is a collection data type which have pairs of **keys** and **values**.
- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.
- Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values In Dictionary

- An empty dictionary without any items is written with just two curly braces, like this: {}.
- To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print("dict['Name']: ", dict['Name'])

print("dict['Age']: ", dict['Age'])
```

Built-in Dictionary Function and Methods

Python includes the following dictionary functions:-

Function	Description
cmp(dict1,dict2)	Compares elements of both dict
len(dict)	Gives the total length of the dictionary. This would be equal to the number of items in the dictionary
str()	Produces a printable string representation of a dictionary
type(variable)	Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type

Cont…

Python includes the following dictionary methods:

Method	Description
dict.clear()	Removes all elements of dictionary dict
dict.copy()	Returns a shallow copy of dictionary dict
dict.fromkeys()	Create a new dictionary with keys from seq and values set to value.
dict.get(key,default=None)	For key key, returns value or default if key not in dictionary
dict.has_key(key)	Returns true if key in dictionary dict, false otherwise
dict.values()	Returns list of dictionary dict’s values
dict.items()	Returns a list of dict's (key, value) tuple pairs
dict.keys()	Returns list of dictionary dict's keys

FUNCTIONS

- A function is a block of organized, reusable code that is used to perform a single, related action.
- As you already know, Python gives us many built-in functions such as print(), but we can also create our own functions. These functions are called **user-defined** functions

Defining a function

- Begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters should be placed within these parentheses. We can also define parameters inside these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Calling a function

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

Function Arguments

A Python function can be called by using the following types of formal arguments:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

The return statement

- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return **None**.

Scope of Variables

There are two basic scopes of variables in Python:

- Global variables
- Local variables

Global Vs. Local Variables

- Local variables

  These are variables that are defined inside a function body.

- Global variables.

  These are variables which are declared outside a function.

OBJECT ORIENTED PROGRAMMING

- Object oriented programming is a very popular paradigm of programming, where objects are created using classes, which are actually the focal point of OOP .
- The class describes what the object will be, but is separate from the object itself.

Overview of OOP Terminologies

- Class
- Object
- Attribute
- Method
- Constructor
- Inheritance
- Polymorphism

Destroying Objects(Garbage Collection)

- Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space.
- The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection.

.

Methods Overriding

You can always override your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

PAUSE

THANK YOU