Week 1: Python Tutorial

Econ 590: Special Topics in Economics

Anuar Assamidanov

Cal State Fullerton

January 25, 2022

Agenda

Today's topics

- Basic Python Data Types
- Lists
- Dictionaries
- Conditionals
- for Loops
- Comprehensions

Basic Python Data Types

Hat Tips

This lecture is inspired heavily by notes and slides created by

- *Python Programming for Data Science* by Tomas Beuzen
- geeksforgeeks.org
- Matt Woerman, University of Massachusetts Amherst

Many thanks to them for generously making their materials available online for all!

Python Variables

- Python is not "statically typed".
  - A variable is created the moment we first assign a value to it
  - A variable is a name given to a memory location
  - The value stored in a variable can be changed during program execution
  - A variable is only a name given to a memory location
- Rules for creating variables in Python:
  - A variable name must start with a letter or the underscore character.
  - A variable name cannot start with a number.
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
  - Variable names are case-sensitive (name, Name and NAME are three different variables

Code example

Everything is an object, and every object has a name and value.

```
1  # An integer assignment
2  age = 45
3  # A floating point
4  salary = 1456.8
5  # A string
6  name = "John"
7
8  print(age)
9  print(salary)
10 print(name)
11
12 <45>
13 <1456.8>
14 <"John">
```

Object Basics

We can re-declare the python variable once we have declared the variable already.

```
# We can re-declare the python variable once we have declared the variable already.
# declaring the var
Number = 100

# display
print("Before declare: ", Number)

# re-declare the var
Number = 120.3

print("After re-declare:", Number)

<Before declare:  100>
<After re-declare: 120.3>
```

Object Basics

How does + operator work with variables?

```
a = 10
b = 20
print(a+b)

a = "Mastering"
b = "Metrics"
print(a+b)

<30>
<"MasteringMetrics">
```

Python Data Types

- Data types are the classification or categorization of data items.
- It represents the kind of value that tells what operations can be performed on a particular data.
- Following are the standard or built-in data type of Python:
  - Numeric
  - Sequence Type
  - Boolean
  - Set (Lists, Strings, Tuple)
  - Dictionary

Numeric

Numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

```
# Python program to demonstrate numeric value
a = 5
print("Type of a: ", type(a))

b = 5.0
print("\nType of b: ", type(b))

c = 2 + 4j
print("\nType of c: ", type(c))

<Type of a:  <class 'int'>>
<Type of b:  <class 'float'>>
<Type of c:  <class 'complex'>>
```

String

Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote.

```
# Creating a String with single Quotes
String1 = 'Welcome to the Econ590'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String with double Quotes
String1 = "I'm a Economist"
print("\nString with the use of Double Quotes: ")
print(String1)
print(type(String1))

<String with the use of Single Quotes:>
<Welcome to the Econ590>

<String with the use of Double Quotes:>
<I'm a Economist>
<class 'str'>
```

String

Individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

```
# Creating a String with single Quotes
String1 = "Machine Learning"
# Printing First character
print("\nFirst character of String is: ")
print(String1[0])
# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])

<First character of String is:>
<M>

<Last character of String is:>
<g>
```

Lists

List

Lists are just like the arrays, declared in other languages which is a ordered collection of data. Lists in Python can be created by just placing the sequence inside the square brackets [ ].

```
# Creating a List with the use of a String
List = ['Machine Learning']
print("\nList with the use of String: ")
print(List)
# Creating a List with the use of multiple values
List = ["Econ", "Economics", "Econometrics"]
print("\nList containing multiple values: ")
print(List)

<List with the use of String:>
<['Machine Learning']>

<List containing multiple values:>
<['Econ', 'Economics', 'Econometrics']>
```

List Indexing

```
List = ["Machine", "Learning", "Engineering"]

# accessing a element from the list using index number
print("Accessing element from the list")
print(List[0])

# accessing a element using negative indexing
print("Accessing element using negative indexing")

# print the last element of list
print(List[-1])
# print the third last element of list
print(List[-3])

<Accessing element from the list>
<'Machine'>

<Accessing element using negative indexing>
<'Engineering'>
<'Machine'>
```

Boolean

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). Note – True and False with capital 'T' and 'F' are valid booleans otherwise python will throw an error.

```
1 print(type(True))
2 print(type(False))
3
4 <class 'bool'>
5 <class 'bool'>
```

Dictionaries

Dictionary

Dictionary in Python is an unordered collection of data values, used to store data values like a map. Unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

```
Dict = {}
print(Dict)

# Creating a Dictionary with Integer Keys
Dict = {1: 'Machine Learning', 2: 'Stat', 3: 'Regression'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

<{}>

<Dictionary with the use of Integer Keys:>
<{1: 'Machine Learning', 2: 'Stat', 3: 'Regression'}>
```

Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets.

```
# Creating a Dictionary
Dict = {1: 'Peter', 'name': 'Matt', 3: 'Adam'}

# accessing a element using key
print("Accessing a element using key:")
print(Dict['name'])

<Accessing a element using key:>
<Matt>
```

Conditionals

if statement

If the simple code of block is to be performed if the condition holds true then if statement is used. Here the condition mentioned holds true then the code of block runs otherwise not.

```
if (condition):
    # Executes this block if condition is true
```

```
# if statement example
if 10 > 5:
    print("10 greater than 5")
print("Program ended")

<10 greater than 5>
<Program ended>
```

if..else Statement

In conditional if Statement the additional block of code is merged as else statement which is performed when if condition is false.

```
if (condition):
    # Executes this block if condition is true
else:
    # Executes this block if condition is false
```

```
# if..else statement example
x = 3
if x == 4:
    print("Yes")
else:
    print("No")

<No>
```

if-elif Statement

The if-elif statement is shoutcut of if..else chain. While using if-elif statement at the end else block is added which is performed if none of the above if-elif statement is true.

```
if (condition):
    statement
elif (condition):
    statement
.
else:
    statement
```

```
letter = "A"
if letter == B:
    print("letter is B")
elif letter == "A":
    print("letter is A")
else:
    print("letter isn't A, B or C")
<letter is A>
```

for Loops

for Loop

Python For loops is used for sequential traversal i.e.

- it is used for iterating over an iterable like string, tuple, list, etc.
- It falls under the category of definite iteration. Definite iterations mean the number of repetitions is specified explicitly in advance
- There is “for in” loop which is similar to for each loop in other languages.

```
1 for var in iterable:
2     # statements
```

for Loop

```
# Python program to illustrate iterating over a list
print("List Iteration")
l = ["Machine Learning", "Economists"]
for i in l:
    print(i)

# Iterating over a String
print("\nString Iteration")
s = "Econ"
for i in s:
    print(i)
```

```
<List Iteration>
<Machine Learning>
<Economists>

String Iteration
E
c
o
n
```

Loop Control Statements

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.

```
# Prints all letters except 'e' and 'o'
for letter in 'economics':
    if letter == 'e' or letter == 'o':
        continue
    print('Current Letter :', letter)

<Current Letter : c>
<Current Letter : n>
<Current Letter : m>
<Current Letter : i>
<Current Letter : c>
<Current Letter : s>
```

range() function

Python range() is a built-in function that is used when a user needs to perform an action a specific number of times.

- start: integer starting from which the sequence of integers is to be returned
- stop: integer before which the sequence of integers is to be returned. The range of integers end at stop – 1
- step: integer value which determines the increment between each integer in the sequence

```
1  printing a number
2  for i in range(10):
3      print(i, end=" ")
4  print()
5
6  # performing sum of first 10 numbers
7  sum = 0
8  for i in range(1, 10):
9      sum = sum + i
10 print("Sum of first 10 numbers :", sum)
11
12 <0 1 2 3 4 5 6 7 8 9>
13 <Sum of first 10 numbers : 45>
```

List Comprehensions

List Comprehension

Python is renowned for encouraging developers and programmers to write efficient, easy-to-understand, and almost as simple-to-read code..

- It is a single line of code to construct powerful functionality
- List comprehensions are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc.
- A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

```
1 newList = [ ]
2 for i in oldList:
3     newList.append(i)
4
5 newList = [expression(element) for element in oldList if condition ]
```

List Comprehension

Below is the implementation of the traditional approach to iterate through a list, string, tuple, etc. Now list comprehension does the same task and also makes the program more simple

```
# Empty list
List = []
# Traditional approach of iterating
for character in 'Machine Learning!':
    List.append(character)
print(List)

<['M', 'a', 'c', 'h', 'i', 'n', 'e', ' ', 'L', 'e', 'a', 'r', 'n', 'i', 'n', 'g', '!']>
```

```
List = [character for character in 'Machine Learning!']
print(List)

<['M', 'a', 'c', 'h', 'i', 'n', 'e', ' ', 'L', 'e', 'a', 'r', 'n', 'i', 'n', 'g', '!']>
```

More Examples – List Comprehension

```
numbers = []
for i in range(1, 6):
    numbers.append(i*10)
print(numbers)

numbers= [i*10 for i in range(1,6)]
print(numbers)

<[10, 20, 30, 40, 50]>
```

```
squares = [n**2 for n in range(1, 11) if n%2==0]
print(squares)

words = ['hello', 'goodbye', 'the', 'antidisestablishmentarianism']
word_lengths = {word:len(word) for word in words} # dictionary comprehension
print(word_lengths)

<[4, 16, 36, 64, 100]>
<{'hello': 5, 'goodbye': 7, 'the': 3, 'antidisestablishmentarianism': 28}>
```