Welcome to

Python and Selenium WebDriver

Part I: Python

Part II: Selenium WebDriver

Introduction To The Course

- What you will gain:
  - ❖ Python knowledge and then Selenium WebDriver
  - ❖ Enough Python skills to write automated web tests
  - ❖ You will be able to read any Python code
  - ❖ Build a test framework while learning

Introduction To The Course – Cont.

- Who is this course for:
  - Anyone looking to learn web automation using Python and WebDriver
  - Manual testers looking to advance their career into Automation
  - Software Testing job seekers who look to increase their chances
  - Any one with a website that like to write tests for his/her website
  - …

Introduction To The Course – Cont.

- Why take the course
  - Say “Built a Framework” on your resume
  - Be able to apply for Automation jobs
  - Dramatically Increase your chances of getting manual QA job
  - Move from manual to automation tester
  - If you already know selenium with java or other language quickly learn it in python

Introduction To The Course – Cont.

- What you need to know before starting
  - Navigating within your computer/system
  - Installing and running programs
  - Very basic HTML

Part I: PYTHON

Introduction To Python

- General purpose high level language
- Both Scripting language and Programming language
- Easy to read and user friendly
- Interpreted language as opposed to complied language
- Runs in most OS (Mac 10x, Windows, Unix, …)
- Object Oriented Programming (OOP)

Introduction To Python – Cont.

- Free
- Fast to develop
- Portable (no change to code needed)
- Wide variety of libraries
- One of the most popular languages
- Example applications:
  - YouTube, Instagram, Dropbox, Spotify(desktop)

Tools and Installation

Tools and Installation

- Python 2.7
  - Windows: will need to install
    - Go to www.ptyon.org and install
  - Mac: comes preinstalled
    - To verify, on the command line do $ which python
      - If no result then you don't have python need to install
      - If "usr/bin/" or similar is the result then you have it
  - Unix: depends what distribution

Tools and Installation

- Pip
  - Install pip
  - Use pip to install iphyton
  - Use pip to install selenium

Tools and Installation – Cont.

- Editor or IDE
  - PyCharm
  - Sublime
  - Eclipse
  - Notepad++

Variables

Variables

- Variables store data
- Help use save values throughout the program
- Using variable enable us to change one place
- and apply to entire program
- Data type of variable does not need to be declared
- Variable can store any data type

Variables – Cont.

- Variable value can change
- Use the assignment operator (=)
- Ex: variable assignment

  >>> x = 20

  >>> _my_var = 'this is the value'
- Everywhere in the program x represents the integer 20

Variables – Cont.

- Variable names have few rules
  - ❖ Must start with letters (upper or lower case)
  - ❖ Or must start with underscore (_)
  - ❖ Other than first character, the rest can be letters, numbers, or underscore
  - ❖ Pep-8 is guideline (https://www.python.org/dev/peps/pep-0008/)
  - ❖ Can not use reserved words for variable name:
    - Ex: print, len, for, if, rand,….

Variables – Cont.

- Reserved Words

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

```
>>> import keyword
>>> keyword.kwlist
```

Variables – cont.

- Variables do not go inside quotes
- If variables are inside quotes, it's a string not a variable
- Ex:

```
>>> car = 'BMW'
>>> print car
>>>BMW
>>> print 'car'
>>>car
```

Variables – Cont.

- Can re-assign variables
- Ex:

```
>>> lunch = 'burgers'
>>> print lunch
>>> burgers
>>> lunch = 'pasta'
>>> print lunch
>>> pasta
```

Variables –Cont.

- Multiple assignment
- Can assign single value to multiple variables
- Ex:

```
>>> my_var1 = my_var2 = my_var3 = 500
```

- Can assign multiple variable to multiple values in one line
- Ex:

```
>>> car1, car2, car3 = 'Honda',  'Toyota', 'BMW'
```

Data Types

Data Types

- Python has many datatypes
- Also referred as Built-in types
  - ❖ Few examples:
    - Numeric types (integers, floats…)
    - Sequence types (strings, lists, tuple,….)
    - Mapping type (dictionaries)
    - Booleans (True, False)
    - And more ….
- Will use most of the above types in Automation

INTEGERS

- Integers (int) are numeric datatype
- Integers are numbers without decimal
- Can be negative, positive, or zero
- Ex: 1 - 23 100 -45
- Can convert strings to integers if compatible
- int(z) →Converts 'z' into an integer (type casting)
- Ex:

```
>>> int('5')
>>> 5
```

Integers – Cont.

- Operations on integers

– Addition (+)
– Subtraction (-)
– Multiplication (*)
– Quotient (division) (/)
– Floored quotient (//)
– Modulus (%)
– abs(x)
– float(x)
– pow(x,y)

Integers – Cont.

- Operator Precedence
- 5 * 2 + 4 → is it 14 or 40
- High school math tell us multiplication has precedence over addition and …..
- So 5*2 gets evaluated first then 4 is added.
- "Please Excuse My Dear Aunt Sally" easy way to remember precedence.
- Parenthesis, Exponents, Multiplication, Division, Addition, Subtraction
- Use parenthesis to avoid confusion and having to remember

FLOAT

- Floats (floating point numbers)
- Numbers with decimal points
- Ex:

  1.0 , 3.4 , -10.5 , 0.6
- 5 and 5.0 are different to python (5 is an int and 5.0 is a float)
- Operation on floating numbers will result in floating numbers

Floats – Cont.

- There are operations that can be applied to floats
- Ex:

```
>>> round(4.3)
>>>4.0
>>>round(4.6)
>>>5.0
```

- Some methods are in math module and need to be imported

```
>>> math.floor(4.8)        >>> math.ceil(4.8)
>>> 4.0                    >>> 5.0
```

STRINGS

- Strings are sequence of one character data
- This sentence is a string
- '5' is a string
- In python string is represented within quotes
- Single quotes and double quotes do not matter
- Keep the sequence of quotes correct(be consistent)
- If double quotes used outside use single inside
  - "The teacher said 'HW is due' tomorrow" ✓
  - "The teacher said "HW is due" tomorrow" ✘

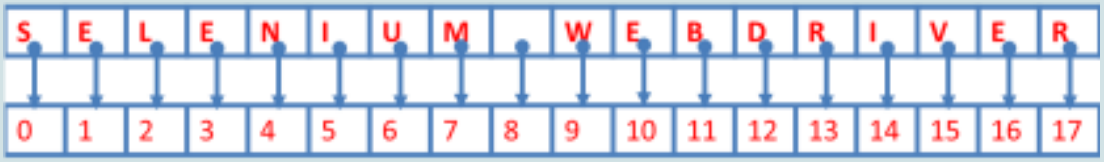
String - Cont.

- **Slicing**: is taking substring of a sting
- Index number: is the location of a character in a string (position)
- Indexing is one of the concepts you will use most as an automation engineer

  >>> my_string = 'SELENIUM WEBDRIVER'
- Indexing starts count from 0 if counting left to right
- Or start from -1 if counting right to left
- index 0 of my_string is 'S' and index 1 is 'E'
- Index -1 of my_string is 'R' and index -2 is 'E'

Strings – Cont.

- The syntax for slicing is variable_name[start index : finish index]
- It does not include the last index
- Ex: >>> my_string = "SELENIUM WEBDRIVER"



>>> my_string[0]
>>> 'S'
>>> my_string[7]
>>> 'M'

>>> my_string[0:]
>>> 'SELENIUM WEBDRIVER'
>>> my_string[9:12]
>>> 'WEB'

Stings – Cont.

- String Methods: are operations that manipulate strings
- There are several strings methods
- Ex:

```
>>> abc=“ Hello World ”
>>> abc.upper()
>>> “ HELLO WORLD ”

>>> abc.lower()
>>> “ hello world ”

>>>abc.strip()
>>> “Hello World”

>>> abc.split()
>>> [‘Hello’, ‘World’]

>>> abc.count(‘l’)
>>> 3

>>> abc.swapcase()
>>> “ hELLO wORLD “

>>> len(str)
>>> 11
```

- https://docs.python.org/2/library/stdtypes.html

Strings – Cont.

- String Formatting:
  - You will use string formatting frequently in automation
  - Specially in reporting and displaying error messages
  - Place holders in strings are %s, %d, %f
  - %s → string
  - %d →integers
  - %f →float
- Ex:

```
>>> ap = ‘Oakland’
>>> my_string = “I am flying to %s airport” % ap
>>> print(my_string)
>>> I am flying to Oakland airport
```

Strings – Cont.

- Python 3.x has different string formatting
- The % way will go away eventually
- Ex:

```
>>> my_var = 'first {} and second {}' .format(44, 'abc')
>>> print my_var
>>> first 44 and second abc
>>> my_var = 'first {1} and second {0}' .format(44, 'abc')
>>> print my_var
>>> first abc and second 44
```

LISTS

- Lists are a mutable (changeable) sequenced data type
- Called "Arrays" in most languages
- List allow us to pack lots of information in one variable
- List start and finish in square brackets
- Each element separated by comma
- Ex. ['I am a string', 5, 'QA', 7.9, '5.7']
- Lists can contain several data types

Lists – Cont.

- Indexing apply for lists as well
- The index number refer to the element at the indexed position
- Again index start from 0 going left to right and from -1 going right to left
- Ex:

```
>>> my_list = ['orange', '2lb', '$5.5', 10]
```

- The item in index 0 is the string 'orange'
- The item in index -1 is the integer 10

Lists – Cont.

- Slicing a list also apply the same way as slicing string
- Ex:

```
>>>my_list = ['car', 'house', 'boat', 'plane']
>>>x = my_list[1:]
>>>print x
>>>['house', 'boat', 'plane']
>>>y = my_list[1:3]
>>>print y
>>>['house, 'boat']
```

List - Continued

- Methods:

  Just like string methods there are also several methods for lists

- Ex:

```
>>> pc = ['Dell', 'HP', 'Toshiba']
>>> len(pc)
>>> 3
>>> pc.append('Apple')
>>> print pc
>>> ['Dell', 'HP', 'Toshiba', 'Apple']
```

```
>>>x = pc.pop()
>>>print pc
>>>['Dell', 'HP', 'Toshiba']
>>>print x
>>>'Apple'
>>> pc.remove("HP")
>>>print pc
```

DICTIONARIES

- Not sequenced
- Open and close with braces { }
- Key:Value pair
- Key and value separated by colon {key:value}
- Each key value pair separated by comma
- {key:value, key:value, key:value}

Dictionaries – Cont.

- Ex:

```
>>> NFL = {“Oakland” : “Raiders” , “San Francisco” : “49ers”, “Denver” : “Broncos”}

>>> presidents_age = {“Obama”: 47, “W. Bush”: 54, “Clinton”: 46}
```

- Any data type can be a value
- Fast for python
- Since they are not sequenced, indexing is not available

Dictionaries – Cont.

```
>>> meal = {“breakfast” : “eggs”, “lunch“ : “salad”}
```

- To add an item in a dict

```
>>> dict_name[key] = value
```

- Ex: add “dinner” to the meal dictionary

```
>>> meal[“dinner”] = “steak”
>>> print meal
>>> {“breakast”:”eggs”, ”lunch”:”salad”, “dinner”:”steak”}
```

Dictionaries – Cont.

- Just like strings and lists, dictionaries have methods
- Ex:

```
>>> cars = {“BMW”: “645i”, “Toyota”: “Camry”, “Audi”: “R8”}
>>> cars.values()
>>> [“645i”, “Camry”, “R8”]  * Note the result is a list
>>> cars.keys()
>>> [“BMW”, “Toyota”, “Audi”]
>>> cars.has_key(“Audi”)
>>> True
```

TUPLES

- Tuples are immutable data types (can not change)
- Can store different types of data just like lists do
- Difference from list is they can not change and they start and end with prentices
- Accessing data from tuple is same as from list.
- Indexing start from 0
  - Ex: tuple1[0] → gives first element in the tuple

TUPLES – Cont

- Examples:

```
>>> tuple_a = (1, 2, 3, 4)
>>> tuple_b = (1, 'x', 'z', 5, 66, 'sample')
>>> tuple_c = ('xy',) # a tuple with one element, note the comma
>>> tuple_d = () # and empty tuple
```

TUPLES – Cont.

- There are built-in functions for tuples also

```
>>> len(tuple_a) → gives the number of elements in tuple_a
>>> tuple(list) → converts a list into a tuple
>>> max(tuple_a) → gives the maximum value in tuple_a
>>> cmp(tuple_a, tuple_b) → compares the two tuples
```

Control Flow

Control Flow – Boolean Operations

- Boolean – Another built-in data type in python
- Booleans are : True, False
- Boolean logic (Boolean Operation)
  - AND, OR, NOT
- AND – requires both values to be True for result to be True
- OR – requires one of the values to be True for the result to be true
- NOT – negates the value that follows it
  - Ex:

    not True → False

    not False → True

Boolean – Cont.

X	Y	operation	Result
True	True	X and Y	True
True	False	X and Y	False
True	False	X or Y	True
False	False	X and Y	False
False	True	X or Y	True
True	False	X and not Y	True

Control Flow - Comparisons

- Operators
  - <= less than or equal to
  - >= greater than or equal to
  - == equal to
  - != not equal to
  - and
  - or
  - not

Control Flow – "if – else" statements

- Made decisions based on it a condition is true or false
- If some condition is true do this but if its not, then do this instead
- Code block for each 'if' and 'else' is indented (4 spaces typically)
- Each statement must end with ":"
- Syntax:

```
>>> if <something is true>:
        <then do this>
    else:
        <do something else>
```

if-else statements – Cont.

- Checking multiple conditions use "elif"
- Keep checking until 'true' is found or 'else' is reached
- Syntax:

```
>>> if <something is true>:
        <do this>
    elif <something else is true>:
        <do this>
    elif <something else is true>:
        <do this>
    else:
        <do this if none of the above is true>
```

Control Flow - Loops

- Loops execute actions repeatedly
- Two types of loops in python
  - 'for' loop
  - 'while' loop

FOR loop

- 'for' loop is counting loop
- Need to use iterable object like a list
- The block of code for the 'for' loop is indented
- The 'for' statement must end with ":"
- Syntax:

  for <variable> in <iterable object>:

      Do some action

  – If the iterable object has X number of items the "Do some action" will repeat X times.

for loop – Cont..

- Ex.

```
>>> my_list = [‘houme’, ‘car’, ‘bike’, ‘boat’]
>>> for i in my_list:
        print i
>>> house
>>> car
>>> bike
>>> boat
```

for loop – Cont.

- Use range(a,b) for known number of iteration
- range(start,end)
- Range does not include the end

```
>>> range(1,5)
>>>[1,2,3,4]
```

- Ex: To repeat something 9 times

```
>>> for j in range(1,10):
        <do something>
```

While loop

- execute code repeatedly until a condition is met
- risky to get infinite loop
- the condition must change to False at some point
- “ctr + c” to stop infinite loop in most systems
- python has its own timeout (do not rely on it)

While loop – Cont.

- Syntax:

```
>>> while <some condition is true>:
        <execute this code>
```

- Ex:

```
>>> counter = 0
>>> while counter <= 4:
        counter += 1
        print 'Currently counter is: %d' % counter
>>>  Currently counter is: 1
     Currently counter is: 2
     Currently counter is: 3
     Currently counter is: 4
     Currently counter is: 5
```

"break" and "continue"

- "break" will exit the loop
- "continue" will make the loop go to top and start the next execution
- What if your loop is supposed to execute 1000 times but you found what you need on the second execution?
  - Then 'break' out of the loop

"break" and "continue" – Cont.

- Syntax:
  - >>> for i in my_list:

```
        Do this
        Do this
        Do this
        if <something is true>:
            break
        Do this
        Do this
```

** while looping the list when the if statement is true then the last two statements will not execute. The loop stop looping (it exits the loop).

"break" and "continue" – Cont.

- Syntax:

```
>>> for i in my_list:
        Do this
        Do this
        Do this
        if <something is true>:
            continue
        Do this
        Do this
```

** while looping the list when the if statement is true then the last two statements will not execute. The loop will go to top and start the next

Functions

Functions

- Functions are block of code packaged in one line
- Functions help us avoid repeated code
- Define a function (a task) once and use repeatedly
- Function definition start with the word 'def'
- Syntax:

```
>>> def my_first_function(input parameters):
        <some code here>
        return <something>
```

- The 'return' statement is optional if nothing to return
- Also the input parameters are optional

Functions – Cont.

- Ex:

```
>>> def my_adding_func(a,b):
        total = a + b
        return total
```

- "Calling a function" means using the function
- Ex:

```
>>> my_adding_func(5,10)
>>> total_value = my_adding_func(5,10)
>>> print total_value
>>> 15
```

Functions – Cont.

- >>> my_adding_func(5,10)
- 5 and 10 are the arguments when calling the function
- Arguments take place of the parameters throughout the function
- If function is defined with parameters it must have arguments when called.
- Number or parameters and arguments must match

Exception Handling

- Exceptions are Errors
- We will learn how to handle them
- Several different types of exceptions
- >>> dir(exceptions)
- Ex:
  - TypeError, IOError, DivisionByZeroError
- >>> Usually we can anticipate the errors and handle them

Exception Handling – Cont.

- 'try' and 'except' (try and catch)
- Syntax:

```
>>> try:
        To do this actions and if
        there is an exception then
    except:
        do this
```

- If there is an exception in the try block then the except block is executed
- Program will not fail unless explicitly coded to fail

Exception Handling – Cont.

- Catch specific exception

```
>>> try:
        result = 5/x
    except ZeroDivisionError:
        print "Can not divide by 0 please try again!"
```

Exception Handling – Cont.

- Catch multiple exceptions with one line

```
>>> try:
        <some code here>
    except <ErrorType1, ErrorType2, ErrorType3>:
        print "Something wrong happened!"
```

Exception Handling – Cont.

- Catch multiple exceptions with multiple lines

```
>>> try:
        result = (x+y)/z
    except (ZeroDivisionError, TypeError):
        print "Something wrong happened!"
    except Exception as e:
        raise Exception ('There is an error. The error is: %s' % e)
```

Exception Handling – Cont.

- 'finally' and 'else'

```
>>>  try:
         result = (x+y)/z
     except:
         raise Exception("Something wrong happened!")
     finally:
         print ('Performing some cleanup')
```

Exception Handling – Cont.

- If we want it to fail we raise an exception
- Can raise specific type of exception
- Can raise exception with custom message.
- Can raise exception at anytime.
- Ex:

```
>>> if total_cost != expected_cost:
        raise Exception('Test Fail!!! The total cost does not equal the expected.')
```