Lab: Intro Python

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Getting setup

You can use Google Colaboratory to run Python. Eventually you'll want to run Python on your own computer though. In Week 1, follow along this video to get everything installed to run Python on your home computer.

https://youtu.be/tSRKB1sLsGE

Python Basics

In this module we will be introducing you to Python, covering fundamentals such as variables, control flow, functions, classes, and packages. You should all be familiar with the language R. R shares many similarities with Python, including some syntax, data types, and uses - R and Python are the two most popular languages for data science [source: edx.org]. Because of this, you should be able to easily understand the basics of Python, and in this lab, we will make some references to R.

Variables and basic types

Much like with R, Python can also be used as a calculator:

```python
# Addition
1 + 1
```

```
2
```

```python
# Subtraction
9 - 6
```

```
3
```

```python
# Multiplication
12 * 8
```

```
96
```

```python
# Division
54 / 18
```

```
3.0
```

```python
# Modulo
39 % 8
```

```
7
```

```python
# Brackets
3 * (4 + 5)
```

```
27
```

```python
# Powers - note that unlike R, Python uses "**" to denote powers
4 ** 3
```

```
64
```

Assigning values to variables is easy:

```python
a = 1
print(a)
```

```
1
```

```python
a = 2
b = 3
b = a
print(b)
```

```
2
```

Types of variables in Python include:

- `int`
- `float`
- `string`
- `bool`
- `NoneType`

You can check the type of a variable by using the `type()` function:

```python
print(type(1))
print(type(1.0))
print(type("Hello World!"))
print(type(1 == 1.0))
print(type(None))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
<class 'NoneType'>
```

Shorthand assignments

```python
x = 1
x += 2
print(x)
```

```
3
```

```python
x = 1
x -= 2
print(x)
```

```
-1
```

```python
x = 1
x *= 2
print(x)
```

```
2
```

```python
x = 1
x /= 2
print(x)
```

```
0.5
```

Strings

Much like in R, you can encode strings by using single or double quotation marks:

```python
s1 = "Hello"
```

```python
s2 = 'World!'
```

You can concatenate strings by using the `+` operator:

```python
s1 + " " + s2
```

```
'Hello World!'
```

F-strings are one of the ways you can substitute the values of variables into a string:

```python
f"{s1} {s2}"
```

```
'Hello World!'
```

Logical operators

In Python, the logical operators `and`, `or`, and `not` are used. This is unlike in R, in which the symbols `&`, `|`, and `!` are used.

```python
True and False
```

```
False
```

```python
True or False
```

```
True
```

Converting types

Use the functions `int()`, `float()`, and `str()` to convert values from one type to another:

```python
x = 3
x = str(x)
print(x)
```

```
3
```

Exercises

1. Calculate the value of $5 * (2 + 4^3)$ using Python.
2. In R, you can use the function `is.string()` to determine whether a value is a string. Unfortunately, there isn't a similar function in Python. Can you figure out a way to check whether a value is a string in Python? What about an integer, or a bool?
3. What is the value of `type(type(bool(int("3"))))`?
4. What is the value of `not(True and not False or False and not (True or False))`?

```python
5*(2+4**3)
```

```
330
```

```python
x = "Hello"

print(x == str(x))
```

```
True
```

```
type(type(bool(int("3"))))
```

```
type
```

```
not(True and not False or False and not (True or False))
```

```
False
```

Data Structures

There are four types of built-in data structures available in Python: lists, dictionaries, tuples, and sets.

Lists are very much like R vectors, in which they hold a series of values of any type. Lists in Python can be altered, whether it be by deleting or adding elements, or by editing specific elements inside the list. This feature is known as **mutability**. You will later see that some of the other Python data structures do not have this feature.

```
l = [1,2,3,4,5]
```

Tuples are similar to lists, except that they are **immutable**, i.e. they cannot be altered.

```
t = (1,3)
```

Dictionaries are a type of data structure that stores data in key-value pairs.

Dictionaries are mutable, however, you cannot change the name of a key.

```
wam_dict = {
    "Alice": 87.4,
    "Bob": 77.9,
    "Charlie": 81.3
}
```

Sets are a type of **unordered** data structure that store a collection of **unique** values. You cannot change the specific values inside a set, but you can remove and insert elements.

```python
fruits = {"Apple", "Banana", "Strawberry"}
print(fruits)

fruits.add("Mango")
print(fruits)

fruits.add("Banana")
print(fruits) #This will not return an error, but the duplicate element will not be added

fruits.remove("Apple")
print(fruits)
```

```
{'Apple', 'Strawberry', 'Banana'}
{'Apple', 'Strawberry', 'Banana', 'Mango'}
{'Apple', 'Strawberry', 'Banana', 'Mango'}
{'Strawberry', 'Banana', 'Mango'}
```

Exercises

a) This exercise will get you familiar with Python's built-in list methods.

1. Create an empty list and assign it to the variable `l`.
2. Append the numbers 7, 2, and 11 to the list.
3. Sort the list using the `sort()` method.
4. Insert the number 8 at index 1 using the `insert()` method.
5. Reverse the list using `reverse()`.
6. Call `pop()` on the list twice.
7. Use `remove()` to remove the number 11 from the list.
8. Print the list.
9. Clear the list.

b) This exercise will teach you how to build a dictionary by combining a list and a tuple. This can also be done with two lists or two tuples.

1. Create a tuple containing the values "Circle", "Triangle", and "Square". Call this tuple `shapes`.
2. Create a list containing the values 1, 3, and 4. Call this list `sides`.
3. Combine `shapes` and `sides` using `dict(zip())`. Print the dictionary.
4. Add `"Pentagon": 5` to this dictionary to show that it is mutable. Print the dictionary.

If-else statements

In Python, note the use of `elif` rather than `else if`. Also, note that Python uses indentation (as opposed to braces in R, C++, JavaScript, and other langagues) to denote different blocks of code.

```python
profit = -30

if profit >= 0:
    print("Profitable")
elif profit == 0:
    print("Break even")
else:
    print("Loss")
```

```
Loss
```

For loops

You can use `for` loops in Python to iterate through each value in a list, tuple, or other collection.

```python
actl_core = ["ACTL1101", "ACTL2111", "ACTL2131", "ACTL2102"]

for course in actl_core:
    print(course)
```

```
ACTL1101
ACTL2111
ACTL2131
ACTL2102
```

You can also use `for` loops to iterate through a sequence of numbers by creating the `range()` object.

```python
for i in range(5):
    print(i ** 2)
```

```
0
1
4
9
16
```

While loops

`while` loops can be used to perform the same tasks as `for` loops, however, they tend to be more cumbersome to put together.

```python
i = 0

while i < 5:
    print(i ** 2)
    i += 1
```

```
0
1
4
9
16
```

`while` loops are useful, however, for creating “sentinel” loops, in which a loop runs until a variable reaches a specified value called a sentinel:

```python
scores = [9, 9, 6, 7, 3, 10, 0, 1, 4, 6, 2]
print(sum(scores))

i = 0
pre_fail_score = 0

while scores[i] != 0: # 0 is a sentinel value
    pre_fail_score += scores[i]
    i += 1

print(pre_fail_score)
```

```
57
44
```

Exercises

1. Recreate the FizzBuzz program - a classic task in coding interviews - in Python. FizzBuzz is a children's game in which people in a circle count to 100 - with a twist. Your FizzBuzz program should loop from 1 to 50, and print, on a new line:

- "Fizz!" if the current number is divisible by 3
- "Buzz!" if the current number is divisible by 5
- "FizzBuzz!" if the current number is divisible by both 3 and 5
- Otherwise, print the current number.

2. This exercise will introduce you to the `break` and `continue` statements in Python.

- Create a new list with the numbers `3, 7, -9, 11, 2, -5, 0, 4`.
- Create a variable, `sum_pos`, and set it equal to 0.
- Create a for loop that iterates through each of the numbers in the list.
- If the current number is positive, add it to `sum_pos`.
- If the current number is negative, ignore it and move to the next number using the `continue` statement.
- If the current number is zero, stop the loop using `break`.

Functions

Functions are instantiated using the `def` keyword.

```python
def addOne(x = 0):
    return x + 1

print(addOne(10))
print(addOne()) #Using default arguments
```

```
11
1
```

Exercises

1. Build a function, `factorial(n)`, that takes in one input `n`, and returns `n!`.
2. Build a function, `fibonacci(n)`, that takes in one input `n`, and returns the `n`th Fibonacci number.