Intro to Python

ECE 241 – Data Structures

Fall 2018
What is Python

• “Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms”.

Source: https://docs.python.org/3/tutorial/index.html
Syntax for Python 3*

```python
>>> 3 + 4 - 2
5

>>> 3 * 4
12

>>> 12 / 3
4.0

>>> 10 / 3
3.3333333333333335

>>> 2**7
128

>>> 12 % 5
2

>>> True
True

>>> 4 == 2
False
```

*Python 2 will not be maintained past 2020
Declaring variables

```python
>>> a = 5
>>> b = 7.0
>>> a + b
12.0
>>> c = True
>>> s1 = 'Hello, world!'
>>> s2 = "Another string"
```

Notice:

No type declaration!
No semicolon!

Strings can be declared with single or double quotes, same result.
Strings are list of characters. More on that later.

*The Python Enhancement Proposal (PEP) 484 introduced Type Annotations (see appendix)
Loops - For

>>> for i in range(5):
   ... print(i)
0
1
2
3
4

Notice:
No brackets!
Watch indentation! (4 spaces)
Loops - While

```python
>>> counter = 1
>>> while counter <= 5:
...   print(counter)
...   counter += 1
1
2
3
4
5
```

No Do-While loops
>>> x = int(input('Enter a number: '))
Enter a number: 5

>>> if x < 0:
...    x = 0
...    print('x is negative')

... elif x == 0:
...    print(x)

... else:
...    print(x)

5

No Switch/Case

‘elif’ is short for Else If
Lists

Create a list
Append an element
Insert element at position
Pop an element and remove from list
Sort list
Sort descending
Returns the index of first occurrence of item
Remove first occurrence of item
String/List operations

Create a string
Iterate through its elements
Create another string
Concatenate strings
String repetition
Return element at position i
Get length of string
String slicing
List comprehensions

Create an empty list and append the squares of i

We can accomplish this with list comprehension in one line
Tuples

- Tuples are very similar to **Lists** in the sense that they are heterogeneous, but similar to **Strings** in the sense that they are immutable.

```python
>>> atuple = (34, 'abc', False, 5.03)
>>> atuple
(34, 'abc', False, 5.03)
>>> len(atuple)
4
>>> atuple*2
(34, 'abc', False, 5.03, 34, 'abc', False, 5.03)
>>> atuple[1:3]
('abc', False)
>>> atuple[2] = True
Traceback (most recent call last):
  File "<stdin>" , line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```
Dictionaries

- Dictionaries are unordered lists of ‘key’ : ‘value’ pairs.
- Examples of dictionaries in real life:
  - Phone book (names : phone numbers)
  - Word dictionary (word : meaning)
  - Password file (user : password hash)
  - Gradebook (course : grade)
- In Python, dictionary keys are unique.
Dictionaries

```python
>>> food_allergies = {'banana': 'no', 'peanuts': 'yes', 'milk': 'yes', 'eggs': 'no'}
>>> food_allergies['shellfish'] = 'yes'
>>> food_allergies['pineapple'] = 'no'
>>> food_allergies
{'banana': 'no', 'peanuts': 'yes', 'milk': 'yes', 'eggs': 'no', 'shellfish': 'yes', 'pineapple': 'no'}
>>> food_allergies.keys()
dict_keys(['banana', 'peanuts', 'milk', 'eggs', 'shellfish', 'pineapple'])
>>> food_allergies.values()
dict_values(['no', 'yes', 'yes', 'no', 'yes', 'no'])
>>> del food_allergies['banana']
>>> food_allergies
{'peanuts': 'yes', 'milk': 'yes', 'eggs': 'no', 'shellfish': 'yes', 'pineapple': 'no'}
```
Functions

• Functions provide an abstraction, or a black box, to the programmer.

• In Python, the keyword `def` defines a function, followed by the function `name`, and a list of `parameters`.

• For example, the function `print_to_std_out` prints a string on the screen.

```python
>>> def print_to_std_out(astring):
...     print(astring)
...     ...
>>> print_to_std_out('Hello, world!')
Hello, world!
```
Classes

Template:

```python
#!/usr/bin/env python
#
# Fraction.py

class Fraction:
    # This is the constructor
def __init__(self, num, den):
        self.num = num
        self.den = den

    # Overriding the __str__() standard method
def __str__(self):
        return str(self.num) + '/' + str(self.den)

    # Declaring the show() method
def show(self):
        print(self.num,'/',self.den)
```

Usage:

```python
>>> from Fraction import Fraction
>>> f1 = Fraction(1, 4)
>>> print(f1)
1/4
>>> print(f1.__str__())
1/4
>>> f1.show()
1 / 4
```
More resources

• Python 3 documentation
  https://docs.python.org/3/tutorial/datastructures.html

• Stack Overflow
  https://stackoverflow.com/questions/tagged/python
  (your questions are most likely already answered)
Appendix – PEP 484 – Type Hints

• Python is a dynamic typed language, i.e., variable types are associated with the value, not with the variable itself. For example:

```python
>>> year = 2018
>>> print(year)
2018
>>> year = 'two thousand eighteen'
>>> print(year)
two thousand eighteen
>>> 
```

• Type hints help developers to understand the code and what a variable should be, but it does not alter the program execution. i.e., type hints are not enforced in runtime.
Appendix – PEP 484 – Type Hints

• Examples

Type hint that variable `year` is intended to be of integer type. But it does not prevent us from assigning a string.

One can also use type hints on a function. Here, the input variable `name` is intended to be a string, and the function is intended to return a string.

See PEP 484 for more details