Going From C++ to Python

Aaron Kaloti, VP of CS Tutoring Club in Winter 2019

March 5, 2019

1 Introduction

This is a guide to introductory Python 3 intended for those with a C++ background. It reviews the differences between C++ and Python 3 in introductory concepts such as conditional statements, strings, and functions. It is intended to make our UC Davis Computer Science tutors more comfortable teaching the introductory Python courses – ECS 32A, 32B, and 36A – newly offered by UC Davis, as our school shifts from focusing on C/C++ at the introductory level to Python.

2 (Quickly) Setting Up Python

Don't have Python 3? Here are two solutions:

1. Python 3 is already on the CSIF.
2. Download Python3 from here. You may wish to download Python IDLE, to have a GUI (if you don't prefer using the terminal).

3 Running a Python Program

- Run a Python program like so, using the 'python3' command. (If you're using IDLE, then do Run Module.) **Python is an interpreted language – no compiler needed.**

```
aaron123@ad3.ucdavis.edu@pc25:~$ cat hello-world.py
    def do_stuff():
        print("I did stuff")

    # Call the function we just defined.
    do_stuff()
aaron123@ad3.ucdavis.edu@pc25:~$ python3 hello-world.py
I did stuff
aaron123@ad3.ucdavis.edu@pc25:~$
```

  – Note that we don't do 'python hello-world.py', as on the CSIF, 'python' would run Python 2 instead of Python 3.

- Use Interpreter Mode to try out things in Python.

```
aaron123@ad3.ucdavis.edu@pc25:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> a
3
>>> b = a
>>> b
3
>>> quit()
aaron123@ad3.ucdavis.edu@pc25:~$
```

4 General Differences

- Lines don't end in semi-colons.
- Types of variables and function parameters aren't explicitly specified.
- Python has automatic garbage collection and thus will "free" your no-longer-needed variables for you, so there's no malloc()/free() or new/delete.
- Comments are indicated by instead of //.

5 Numbers and Arithmetic

- +, -, *, and % are the same.
- Unlike in C++, / doesn't truncate in Python when both operands are integers. You must use // to cause truncation.

```
>>> 3 / 2
1.5
>>> 3 // 2
1
>>>
```

- Use ** for exponentiation.

```
>>> 5 ** 2  # 5 squared
25
```

6 Standard Input/Output

- Use print() to print to standard output.
  - For fans of C++'s printf():

    ```
    >>> print("{} says {}".format("Aaron","hi"))
    Aaron says hi
    ```

- Use input() for basic standard input.

  ```
  >>> name = input("Enter your name: ")
  Enter your name: Aaron
  >>> name
  'Aaron'
  >>>
  ```

7 Lists

- Lists in Python are arrays in C++, but you needn't do any special allocation stuff. The major list operations (access, length, splicing, concatenation, modification, append, delete) are demonstrated:
- Define a list.

  ```
  >>> mylist = ['a','b','c','d','e']  # list of characters
  ```

- Access element of a list.

  ```
  >>> mylist[0]  # Python uses zero-based indexing like C++
  'a'
  >>> mylist[4]
  'e'
  ```

- Access element of a list, starting from the back.

  ```
  >>> mylist[-1]  # get first element from back
  'e'
  >>> mylist[-2]
  'd'
  ```

- Get length of a list.

  ```
  >>> len(mylist)  # get length of list
  5
  ```

- Splice a sublist from the list.

```
>>> mylist[0:2]  # splice from index 0 to before index 2
['a', 'b']
>>> mylist[1:4]  # splice from index 1 to before index 4
['b', 'c', 'd']
>>> mylist[2:]  # splice from index 2 to end
['c', 'd', 'e']
>>> mylist[-2:]  # splice from second-to-last element onwards
['d', 'e']
```

- Concatenate two lists.

```
>>> ['t','u','v'] + ['w','x']  # concatenation
['t', 'u', 'v', 'w', 'x']
```

- Modify a specific list element.

```
>>> mylist[2] = 'x'  # modification: replace 'c' with 'x'
>>> mylist
['a', 'b', 'x', 'd', 'e']
```

- Append an element to a list.

```
>>> mylist.append('z')  # append 'z' to back of list
>>> mylist
['a', 'b', 'x', 'd', 'e', 'z']
```

- Delete an element from the list.

```
>>> del mylist[1]  # delete 'b' from list
>>> mylist
['a', 'x', 'd', 'e', 'z']
```

- Get the type of this list.

```
>>> type(mylist)
<class 'list'>
```

8 Strings

- Strings in Python are strings in C++, but Python has no characters (a character is a string of length 1).
- No difference between single quote and double quote.
- Ignoring modification operations, strings and lists have the same operations.
- Define a string.

```
>>> mystr = "aaron kaloti"
```

- Access element of a string.

```
>>> mystr[0]  # again, zero-based indexing
'a'
>>> mystr[-2]  # second-to-last character
't'
```

- Get length of a string.

```
>>> len(mystr)
12
```

- Splice a substring from the string.

```
>>> mystr[:5]  # splice to get my first name
'aaron'
>>> mystr[6:]  # splice to get my last name
'kaloti'
```

- Concatenate two strings.

```
>>> "concat" + "enation"
'concatenation'
```

- Get the type of this string.

```
>>> type(mystr)
<class 'str'>
```

- IMPORTANT: In Python, we call strings "immutable". This means that, unlike with a list, **you can't modify an individual element in a string**. If you want to change a string, you must use concatenation (to create a new string).

```
>>> mystr
'aaron kaloti'
>>> mystr[1] = 'd'
>>> mystr[3] = 'i'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> mystr = mystr[:3] + 'i' + mystr[4:]
>>> mystr
'aarin kaloti'
```

9 Conditional Statements

- If/else statements are the same, besides syntactic differences (no parentheses around the condition, condition ends with a colon, indentation indicates the body of the if/else, and we use "elif" instead of "else if"):

```
>>> x = 8
>>> if x < 0:
...     print('x is negative')
... elif x == 0:
...     print('x is zero')
... else:
...     print('x is positive')
...
x is positive
>>>
```

10 Iteration

- While loops are the same, besides minor syntactic differences:

```
>>> mylist = ['dog','cat','mouse']
>>> i = 0
>>> while i < len(mylist):
...     print(mylist[i])
...     i += 1
...
dog
cat
mouse
>>>
```

- For loop to iterate across a range of values (here, the variable **i** needn't be initialized prior):

```
>>> for i in range(2,6):  # last value (6) isn't included
...     print(i)
...
2
3
4
5
>>> for i in range(3):  # range starts at 0 if only give one value
...     print(i)
...
0
```

```
1
2
>>>
```

- For loop to iterate across the values in a list:

  ```
  >>> people = ['Aaron','Aakash','Matthew']
  >>> for person in people:
  ...     print(person)
  ...
  Aaron
  Aakash
  Matthew
  >>>
  ```

  - Note that when using this syntax, we can't change the values in the list.

    ```
    >>> people = ['Aaron','Aakash','Matthew']
    >>> for person in people:
    ...     person = "Alex"
    ...
    >>> people  # note that the list is unaffected
    ['Aaron', 'Aakash', 'Matthew']
    >>>
    ```

- **break** and **continue** work the same.

11 Functions

- Types of function parameters aren't specified.
- No return type is specified, so a function can return different types of values (or in some cases, no value at all).
- Here is an example to illustrate syntactic differences:

  ```
  >>> def isEven(val):
  ...     if val % 2 == 0:
  ...             return True
  ...     else:
  ...             return False
  ...
  >>> isEven(3)
  False
  >>> isEven(4)
  True
  >>>
  ```

- Default argument values:

```
>>> def returnInput(val=8):
...     return val
...
>>> returnInput(3)
3
>>> returnInput()  # use default argument
8
>>>
```

12 Tuples

13 Dictionaries

14 File Input/Output

15 Command-line arguments

16 Exceptions

17 Classes

- NOTE: User-defined classes shouldn't come up in ECS 32A or 36A, but they do come up briefly in Kurt Eiselt's ECS 32B. (He doesn't cover inheritance, but if a future ECS 32B instructor does, I'll update this guide.)