CSC 271 - Software I: Utilities and Internals

Lecture #8 – An Introduction to Python

Programming in Python

- Python is a general purpose interpreted language. There are two main versions of Python; Python 2 and Python 3. For this course, we will use version 2.7 of the language.
- Our first program:

```
#!/usr/bin/python

print "Hello, World!"
```

Running `Hello.py`

- We can run it from the prompt:

```
SIEGFRIE@panther:~$ python hello.py
Hello, World!
SIEGFRIE@panther:~$
```

- Note that we did not define have to define a main method, return any values, or terminate the statement with a semicolon. It would look cleaner if we had written:

```
#!/usr/bin/python

def main():
        print "Hello, World!"

if __name__ == "__main__":
        main()
```

__name__ *is the name of a system-defined variable that defines the scope in which the script executes.*
The top-level scope is called "__main__".

Variables in Python

- Variables are implicitly defined when they are assigned their first value.
- Python uses dynamic typing, which means that all variables do have a type, but that you cannot declare it.
  - Type is determined at time of assignment.
- Also, data types are dynamically bound to a variable, which means that the data type of a variable changes through execution.
- Variable names are case sensitive.

main.py

```
SIEGFRIE@panther:~$ cat main.py
#!/usr/bin/python

def main():
        x = "4"
        print x

        x = 4
        print x

        x = 4.0
        print x

if __name__ == "__main__":
        main()
SIEGFRIE@panther:~$ python main.py
4
4
4.0
```

Assignment Operator

- The **=** is the assignment operator. It both assigns a value to a variable, and it determines its data type.
- The assignment operator can take multiple values at once.

Example – **hello3.py**

```
SIEGFRIE@panther:~$ cat hello3.py
#!/usr/bin/python

def main():
        a, b = "Hello", 'world!'
        print a, b

if __name__ == "__main__":
        main()
SIEGFRIE@panther:~$ python hello3.py
Hello world!
```

Example – **four.py**

```
SIEGFRIE@panther:~$ cat four.py
#!/usr/bin/python

def main():
        a = b = 4
        print a, b

if __name__ == "__main__":
        main()
SIEGFRIE@panther:~$ python four.py
4 4
SIEGFRIE@panther:~$
```

Strings

- Python does not distinguish single quotes from double quotes.
- Use the one that makes your code most readable.

Example – **yall.py**

```
SIEGFRIE@panther:~$ cat yall.py
#!/usr/bin/python

def main():
        x = "Hey y'all!"
        print x
        x = 'He said: "How are you all doing?"'
        print x

if __name__ == "__main__":
        main()
SIEGFRIE@panther:~$ python yall.py
Hey y'all!
He said: "How are you all doing?"
SIEGFRIE@panther:~$
```

Numerical Data Types

- Python distinguishes integers and floats. Both data types do not have a limit on their size. However, when a Python integer overflows, computational performance suffers quite drastically.
- Floating point precision is limited to 53 bits, which is generally more than enough.

Boolean

- Unlike C and C++, but just as Java, Python has a built-in knowledge of a boolean data type.
- The truth values are True and False, with an upper case initial character.

Scope of Variables

- Variables defined in the main body are called global variables, and are visible in all functions.
- However, global variables cannot be changed from within functions without additional statements.
- Functions defined in a function are visible within that function only.

Example – **`funky.py`**

```
SIEGFRIE@panther:~$ cat funky.py
a = 10

def func():
        a = 5
        print a

func()
print a

SIEGFRIE@panther:~$ python funky.py
5
10
```

Example – **`funky.py`**

```
SIEGFRIE@panther:~$ cat funky2.py
a = 10

def func():
        global a
        a = 5
        print a

func()
print a

SIEGFRIE@panther:~$ python funky2.py
5
5
SIEGFRIE@panther:~$
```

Lists and Sets

- Python does not know the concept of an array. Instead, it has lists.
  - Lists are different from arrays in that a list can contain values of different types.
  - Lists may contain duplications and list elements retain their order.
- Sets cannot contain duplicates, and they are not ordered.

Example – **`lists.py`**

```
SIEGFRIE@panther:~$ cat lists.py
#/usr/bin/python

def main():
        words = []      # create an empty list
        words.append("Hello")
        words.append("World")

        print words[0], words[1]

if __name__  == "__main__":
        main()

SIEGFRIE@panther:~$ python lists.py
Hello World
SIEGFRIE@panther:~$
```

Example – **`sets.py`**

```
SIEGFRIE@panther:~$ cat sets.py
#!/usr/bin/python

def main():
    words = {"apples", 'oranges', "kiwis", "oranges"}
    print words

if __name__ == "__main__":
    main();
SIEGFRIE@panther:~$ python sets.py
set(['kiwis', 'apples', 'oranges'])
SIEGFRIE@panther:~$
```

Dictionaries

- The dictionary datatype contains a list of attribute-value pairs. The attribute is know as the ***key***.

Example – `dict.py`

```
SIEGFRIE@panther:~$ cat dict.py
#!/usr/bin/python

def main():
  words = {
      "aardvark":
         'The aardvark is a medium-sized, burrowing, '
                'nocturnal mammal native to Africa.',
      'apple':
         "A nice piece of fruit"
  }
  print words, "\n"
  print words.keys(), "\n"
  print words['apple']

if __name__ == "__main__":
        main()
```

```
SIEGFRIE@panther:~$ python dict.py
{'aardvark': 'The aardvark is a medium-sized,
burrowing, nocturnal mammal native to Africa.',
'apple': 'A nice piece of fruit'}

['aardvark', 'apple']

A nice piece of fruit
SIEGFRIE@panther:~$
```

Basic Operators

```
SIEGFRIE@panther:~/python$ cat ops.py
#!/usr/bin/python

def main():
        # Numerical addition and subtraction
        print 4 + 4 - 2

        # Numerical multiplication and division
        print 4 * 2 / 3.0

        # Merging two lists
        print ["apples", "oranges"] + ["kiwis", "bananas"]

        # String concatenation
        print "Hello " + "world!"
        # Non-numerical multiplication
        print "a" * 5;
        print [1, 2, 3] * 2;

        # Remainder after division (modulus)
        print -4 % 3
        print 4 % 3

        # power
        print 2 ** 20

if __name__ == "__main__":
        main()
```

```
SIEGFRIE@panther:~/python$ python ops.py
6
2.66666666667
['apples', 'oranges', 'kiwis', 'bananas']
Hello world!
aaaaa
[1, 2, 3, 1, 2, 3]
2
1
1048576
SIEGFRIE@panther:~/python$
```

Conditions

- Syntax:

```
if  <statement is true>:
    <do something>
    …
elif <other statement is true>:
    <do something else>
    …
else:
    <do yet something else>
```

Example: `tod.py`

```
SIEGFRIE@panther:~/python$ cat tod.py
#!/usr/bin/python
def main():
        hour = 12
        if hour < 6:
                tod = "night"
        elif hour < 12:
                tod = "morning"
        elif hour < 18:
                tod = "afternoon"
        else:
                tod = "evening"
        print tod
if __name__ == "__main__":
        main()
SIEGFRIE@panther:~/python$ python tod.py
afternoon
```

Conditions

- Noteworthy:
  - No parentheses around condition
  - Blocks through indentation, not through bracing
  - Don't use '**`else if`**', instead use **`elif`**
- **`and`**, **`or`**, **`not`** conditions are written using English words.
- **`>`**, **`>=`**, **`<=`**, **`!=`** have their common meanings

Example – `dow.py`

```
SIEGFRIE@panther:~/python$ cat dow.py
#~/usr/bin/python\

def main():
        day = "Wednesday"
        if (day == "Friday" and hour > 17) or \
                (day == "Saturday" or day ==
"Sunday"):
                weekend = True
        else:
                weekend = False

        if not weekend:
                print "Go to work"
```

```
if __name__ == "__main__":
        main()
SIEGFRIE@panther:~/python$ python dow.py
Go to work
SIEGFRIE@panther:~/python$
```

`for` Loops

- The for-loop iterates over an iterable sequence.

```
for day in ["sunday", "monday", "tuesday",\
          "wednesday", "thursday","friday",\
          "saturday"]:
       print day
```

`for` Loops

- That also means that the traditional for-loop that you know from C, C++ or Java does not work.
- Alternatively, you would write

```
for number in range(0, 10):
        print number
```

- Prints numbers 0-9
- Note that the variable continues to exist outside the **`for`** statement!

`while` Loops

- Syntax:
- **`While`** *`<condition is True>`*`:`
    `<Some Statements>`
- Example

```
count = 0
while count < 5:
     print count
     count +=1
```

while Loops

- More interesting is that while and for statements support an else clause, too:

```
count = 0
while count < 5:
     print count
     count += 1
else:
     print "Target reached."
```

- The else statement is executed if the loop exists as the result of failing the loop condition.

continue

- The continue statement skips the remainder of the current block and return to the enclosing statement:

```
even = True
for number in range(1, 10):
        even = not even
        if even:
                continue
        print number
```

break

- The break statement ends a loop completely.
- Generally, structured programming practices discourage the use of a break statement.
- However, in some cases, break can be useful to avoid kludgy code.
- If a loop terminate using a break statement, the else component is not executed.

break – An Example

```
count = 0
while count < 2 ** 128:
        count += 1
        if count == 100:
                break
        else:
                print "Target reached"
```

String Formatting

- To format a string in the same style that C lets you, use the regular print statement.

```
for counter in range(10):
        s = ""
        if counter % 2 == 0:
                s += "apples"
        elif counter % 3 == 0:
                s += "kiwis"
        else :
                s += "oranges"

        print "%02d: %s" % (counter, s)
```

String Formatting

```
SIEGFRIE@panther:~/python$ python format.py
00: apples
01: oranges
02: apples
03: kiwis
04: apples
05: oranges
06: apples
07: oranges
08: apples
09: kiwis
SIEGFRIE@panther:~/python$
```

String Formatting

```
SIEGFRIE@panther:~/python$ cat sformat.py
#!/usr/bin/python

myDict={"a": "Letter A",
        "b": "Letter B",
        "c": "Letter C"
}

mySet={'a', 'b', 'c', 'a'}

myList=['a', 'b', 'c', 'd']

print "%s\n%s\n%s" % (myDict, mySet, myList)
SIEGFRIE@panther:~/python$ python sformat.py
{'a': 'Letter A', 'c': 'Letter C', 'b': 'Letter B'}
set(['a', 'c', 'b'])
['a', 'b', 'c', 'd']
SIEGFRIE@panther:~/python$
```

Functions

- Functions are defined using the **`def`** keyword, followed by the name of function and the list or arguments.
- Functions may return a value, but don't have to.
- You do not need to define a return type.

Function - Example

```
SIEGFRIE@panther:~/python$ cat funcs.py
#!/usr/bin/python

def add(a, b):
    """It is good practice to document Python
    functions with a multiline string.  The
    multiline string is delimited by three double-
    quotation marks."""

    return a + b # return the sum of A and B

print add(5, 4)
SIEGFRIE@panther:~/python$ python funcs.py
9
SIEGFRIE@panther:~/python$
```

Example – Computing Min, Max, Avg

```
SIEGFRIE@panther:~/python$ cat minmaxavg.py
#!/usr/bin/python

def minmaxavg(numList):
   max=min=numList[0]
   sum = 0.0

   for n in numList:
        if n > max:
                max = n
        if n < min:
                min = n
        sum += n

   return max, min, sum/len(numList)
```

```
(max, min, avg) = minmaxavg([31, 114, 15, 92, 65,\
                             35, 9])

print "Min: %2d\tMax: %2d\tAvg: %2d" % (min, max, avg)
SIEGFRIE@panther:~/python$ python minmaxavg.py
Min:  9 Max: 114        Avg: 51
SIEGFRIE@panther:~/python$
```

Example – Bubble Sort

```
SIEGFRIE@panther:~/python$ cat bubsort.py
#!/usr/bin/python

def bubblesort(myList):
   for i in range(len(myList)):
      for j in range(i+1, len(myList)):
         if myList[j] < myList[i]:
            myList[j], myList[i] = myList[i],
myList[j]
   return myList

print bubblesort([31, 14, 15, 92, 65, 35, 9])
```

```
SIEGFRIE@panther:~/python$ python bubsort.py
[9, 14, 15, 31, 35, 65, 92]
SIEGFRIE@panther:~/python$
```