CSCE 110 — Programming I
Basics of Python (Part 3): Lists, Functions, Tuples, and For Loops

Dr. Tiffani L. Williams

Department of Computer Science and Engineering
Texas A&M University

Fall 2013
A function is a named block of statements that performs an operation.

To define a function, we use the following syntax:

```python
def func(param_list):
    block
```

When executed, this compound statement creates a new function object and assigns it to the name `func`.

`func` is a valid Python name (think of valid variable names), `param_list` represents zero or more comma-separated parameters, and `block` is an indented block of statements.
# A simple example of how to use functions

def print_msg():
    print "I love Python!"

def iseven(num):
    print num % 2 == 0

print_msg()
iseven(10)
iseven(7)
Listing 2: temperature-converter.py

```python
# Converts the temperature to Celsius or Fahrenheit

def to_fahrenheit(c):
    # Convert celsius to fahrenheit
    return (c * 9.0/5.0) + 32

def to_celsius(f):
    # Convert fahrenheit to celsius
    return (f - 32) * 5.0/9.0

type = raw_input("Convert temperature to Celsius or Fahrenheit (c or f)? ")
if type == 'c':
temperature = int(raw_input("Enter Fahrenheit temperature: "))
celsius = to_celsius(temperature)
print "%d Fahrenheit is %d Celsius." % (temperature, celsius)
else:
temperature = int(raw_input("Enter Celsius temperature: "))
fahrenheit = to_fahrenheit(temperature)
print "%d Celsius is %d Fahrenheit." % (temperature, fahrenheit)
```
# Converts the temperature to Celsius or Fahrenheit. Uses a main function to drive the program.

def to_fahrenheit(c):
    return (c * 9.0/5.0) + 32

def to_celsius(f):
    return (f - 32) * 5.0/9.0

def main():
    type = raw_input("Convert temperature to Celsius or Fahrenheit (c or f)? ")
    if type == 'c':
        temperature = int(raw_input("Enter Fahrenheit temperature: 
          celsius = to_celsius(temperature)
        print "%d Fahrenheit is %d Celsius." % (temperature, celsius)
    else:
        temperature = int(raw_input("Enter Celsius temperature: 
          fahrenheit = to_fahrenheit(temperature)
        print "%d Celsius is %d Fahrenheit." % (temperature, fahrenheit)

# Execution of the program begins here
main()
## Built-in Functions

<table>
<thead>
<tr>
<th>abs()</th>
<th>divmod()</th>
<th>input()</th>
<th>open()</th>
<th>staticmethod()</th>
</tr>
</thead>
<tbody>
<tr>
<td>all()</td>
<td>enumerate()</td>
<td>int()</td>
<td>ord()</td>
<td>str()</td>
</tr>
<tr>
<td>any()</td>
<td>eval()</td>
<td>isinstance()</td>
<td>pow()</td>
<td>sum()</td>
</tr>
<tr>
<td>basestring()</td>
<td>execfile()</td>
<td>issubclass()</td>
<td>print()</td>
<td>super()</td>
</tr>
<tr>
<td>bin()</td>
<td>file()</td>
<td>iter()</td>
<td>property()</td>
<td>tuple()</td>
</tr>
<tr>
<td>bool()</td>
<td>filter()</td>
<td>len()</td>
<td>range()</td>
<td>type()</td>
</tr>
<tr>
<td>bytearray()</td>
<td>float()</td>
<td>list()</td>
<td>raw_input()</td>
<td>unichr()</td>
</tr>
<tr>
<td>callable()</td>
<td>format()</td>
<td>locals()</td>
<td>reduce()</td>
<td>unicode()</td>
</tr>
<tr>
<td>chr()</td>
<td>frozenset()</td>
<td>long()</td>
<td>reload()</td>
<td>vars()</td>
</tr>
<tr>
<td>classmethod()</td>
<td>getattr()</td>
<td>map()</td>
<td>repr()</td>
<td>xrange()</td>
</tr>
<tr>
<td>cmp()</td>
<td>globals()</td>
<td>max()</td>
<td>reversed()</td>
<td>zip()</td>
</tr>
<tr>
<td>compile()</td>
<td>hasattrib()</td>
<td>memoryview()</td>
<td>round()</td>
<td><strong>import</strong>()</td>
</tr>
<tr>
<td>complex()</td>
<td>hash()</td>
<td>min()</td>
<td>set()</td>
<td>apply()</td>
</tr>
<tr>
<td>delattr()</td>
<td>help()</td>
<td>next()</td>
<td>setattr()</td>
<td>buffer()</td>
</tr>
<tr>
<td>dict()</td>
<td>hex()</td>
<td>object()</td>
<td>slice()</td>
<td>coerce()</td>
</tr>
<tr>
<td>dir()</td>
<td>id()</td>
<td>oct()</td>
<td>sorted()</td>
<td>intern()</td>
</tr>
</tbody>
</table>
Listing 4: local-global-variables.py

```python
# An example showing the difference between local and global variables.

v = 15  # v is a global variable

def f1():
    v = 17  # v is a local variable
    print 'v(f1):', v
    v = v + 1
    print 'v(f1):', v

def f2():
    print 'v(f2):', v  # since v is not local, global version is used

f1()
f2()
print 'v:', v  # references global version of v
```

# An example showing the difference between local and global variables.

v = 15  # v is a global variable

def f1():
    v = 17  # v is a local variable
    print 'v(f1):', v
    v = v + 1
    print 'v(f1):', v

def f2():
    print 'v(f2):', v  # since v is not local, global version is used

f1()
f2()
print 'v:', v  # references global version of v
# This program has an error. Find it and explain why it is in fact an error.

```python
v = 15

def f1():
    v = 17
    print 'v(f1):', v

def f2():
    v = v + 10
    print 'v(f2):', v

f1()
f2()
print 'v:', v
```

Listing 5: local-global-variables2.py
Lists

- Ordered collection of data.
  - Example: `some_data = ['dog', 78, 87.0, 'gorilla']`
  - Elements can be of different types (heterogeneous)
  - Can have a mixture of strings, ints, floats, lists, etc.

- Composed of elements that can be accessed by indexing

- Can create sublists by specifying an index range
  - This is accomplished with the slicing operator `[:]` or `[::]`

- You can change individual elements directly ("mutable")
  - Unlike strings, each element in a list can be modified

- List creation operator `[]`, elements of the list are separated by commas
List Examples (Python Shell)

```python
>>> aList = [1, 2, 3, 4]  # list creation
>>> aList
[1, 2, 3, 4]

>>> aList[0]  # indexing individual elements
1

>>> aList[2:]  # creating sublist
[3, 4]

>>> aList[:3]  # creating sublist
[1, 2, 3]

>>> aList[1] = 5  # mutable
>>> aList
[1, 5, 3, 4]
```
List Examples (Python Shell)

```python
>>> aList = [1, 2, 3, 4]  # list creation
>>> aList
[1, 2, 3, 4]

>>> aList[0]  # indexing individual elements
1

>>> aList[2:]  # creating sublist
[3, 4]

>>> aList[:3]  # creating sublist
[1, 2, 3]

>>> aList[1] = 5  # mutable
>>> aList
[1, 5, 3, 4]
```
for Loop

```python
for iter_var in iterable:
    suite_to_repeat
```

Objects that are iterable include strings, lists, and tuples.
for Loop Examples

Listing 6: for-example.py

```python
1 for eachletter in "Names":
2     print "current letter:", eachletter
```

Listing 7: for-example2.py

```python
1 name_list = ['Walter', "Nicole", 'Steven']  # iterating over a list
2 for each_name in name_list:
3     print each_name, "Smith"
```
while vs for loops

Listing 8: while-vs-for.py

```python
# Shows the difference between while and for loops by
# printing the numbers from 1 to 5.

print "while loop: Printing the numbers from 1 to 5."

i = 1
while i < 6:  # could also write while i <= 5:
    print i
    i += 1

print "\nfor loop: Printing the numbers from 1 to 5."

for i in range(1,6):  # range(1,6) creates the list [1, 2, 3, 4, 5]
    print i
```
Integer to English Conversion

Given an integer value, return a string with the equivalent English text of each digit. For example, an input of 89 results in “eight-nine” being returned.

For an extra challenge, return English text with proper usage, i.e., “eighty-nine.” For this problem, restrict values to be between 0 and 1,000.
# A program that converts an integer between 0 and 1,000 to its English equivalent.
# Returns the English equivalent of the numbers 0 to 9.
def convert(digit):
    if digit == '0':
        name = 'zero'
    elif digit == '1':
        name = 'one'
    elif digit == '2':
        name = 'two'
    elif digit == '3':
        name = 'three'
    elif digit == '4':
        name = 'four'
    elif digit == '5':
        name = 'five'
    elif digit == '6':
        name = 'six'
    elif digit == '7':
        name = 'seven'
    elif digit == '8':
        name = 'eight'
    else:
        name = 'nine'

    return name
# Obtains the user’s input and farms the work.

def main():

    result = ''
    number = raw_input('Please enter an integer between 0 and 1000: ')

    for digit in number:
        result += convert(digit) + '-'

    print '%s is %s.' % (number, result[:len(result)-1])

# Execution begins here.
main()
# A program that converts an integer between 0 and 1,000 to its English equivalent.
# Easy version.

# Returns the English equivalent of the numbers 0 to 9.
def convert(digit):
    name = ['zero', 'one', 'two', 'three', 'four', 'five',
            'six', 'seven', 'eight', 'nine']
    return name[int(digit)]

# Obtains the user’s input and farms the work.
def main():
    result = []
    number = raw_input('Please enter an integer between 0 and 1000: ')

    for digit in number:
        result += [convert(digit)]

    result = '-'.join(result)

    print '%s is %s.' % (number, result)

main()
Tuples

- Tuples are similar to lists except for one important difference. Unlike lists, tuples are immutable.
  - Example: `some_data = ('dog', 78, 87.1, 'gorilla')`
  - An element in a tuple cannot be changed. In that sense, both strings and tuples share the immutability criterion.
  - Reason for immutability: you don’t want variable's contents to be accidentally overwritten.

- Tuple creation operator `()`, elements of the list are separated by commas
>>> aTuple = ('robots', 77, 93, 'try')  # tuple creation
>>> aTuple
('robots', 77, 93, 'try')

>>> aTuple[:3]  # creating subtuples
('robots', 77, 93)

>>> aTuple[1] = 5  # immutable

Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
TypeError: 'tuple' object does not support item assignment
1 >>> aList = [1, 2, 3]            # list creation
2 >>> aList + [4, 5]               # list concatenation
3 [1, 2, 3, 4, 5]

4 >>> aTuple = ('four', 'five')   # tuple creation
5 >>> aTuple + ('six')            # immutable
6 Traceback (most recent call last):
7     File "<string>", line 1, in <fragment>
8     TypeError: can only concatenate tuple (not "str") to tuple

9 >>> aList                    # print aList
10 [1, 2, 3]
11 >>> aTuple                   # print aTuple
12 ('four', 'five')
13 >>> aList + [aTuple]         # concatenate list and tuple as a list
14 [1, 2, 3, ('four', 'five')]
15
16 >>> aList + aTuple            # immutable
17 Traceback (most recent call last):
18     File "<string>", line 1, in <fragment>
19     TypeError: can only concatenate list (not "tuple") to list