CSCE 110 — Programming I
Basics of Python: Lists, Tuples, and Functions

Dr. Tiffani L. Williams

Department of Computer Science and Engineering
Texas A&M University

Fall 2011
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
>>> 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]
```
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
for Loop

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

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

Listing 1: for-example.py

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

Listing 2: for-example2.py

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

Listing 3: while-vs-for.py

# 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
Concatenating Lists or Tuples: Examples

```python
>>> aList = [1, 2, 3]  # list creation
>>> aList + [4, 5]    # list concatenation
[1, 2, 3, 4, 5]

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

>>> aList  # print aList
[1, 2, 3]
>>> aTuple  # print aTuple
('four', 'five')
>>> aList + [aTuple]  # concatenate list and tuple as a list
[1, 2, 3, ('four', 'five')]

>>> aList + aTuple  # immutable
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
TypeError: can only concatenate list (not "tuple") to list
```
Defining a Function

- 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.
Function Examples (1)

Listing 4: function-example.py

```python
# 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)
```
# 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)
Listing 6: temperature-converter2.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

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()
```
1. 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.

2. Prime number. Write a program that asks a person for a number and returns whether the value is prime or not.