Accelerating Information Technology Innovation

http://aiti.mit.edu

Nigeria Summer 2011
Lecture 2 – Data Structures and Control Structures
Agenda

• Data Structures (Tuples, Dictionaries)
• Learn about:
  – If...Else
  – Loops
    • While
    • For
    • Nested Loops
• Functions
Tuples (): Introduction

• Essentially an **immutable** list
  – **CANNOT** change list items
  – Form: `tuple=(1, 2, 3, 4,...)`

• We saw an example of this earlier:
  – `team_3_tuple=('Peju', 'Ibrahim', 'Sunday', 'Gbadebo')`
Tuples: Manipulation

• **NOTICE:**
  - `tuple[0] = 'A'` returns an **error** because tuples are not mutable

• There are **some** ways around this
  - Make new tuple and **add** part of existing tuple
    
    ```python
    tuple = ('A',) + tuple[1:]
    
    New Tuple: ('A', 'b', 'c', 'd', 'e')
    ```
Lists and Tuples: Limitations

• Suppose 50 students in each AITI Team
• How do we check that Akin is in the team? Are there any shortcuts?
  – Sorted lists can help, BUT
  – Costly to insert new elements into sorted lists

• A different solution: dictionaries
Dictionaries

- An unordered collection of (key, value) pairs
- (key, value) pairs are mappings
  - key: something you know
  - value: something you want to know that is related to the key
- Key and value can be objects of any type
Dictionaries: Initialization

• Initialization (maps students to teams):

```python
student_team = {'Malik':'team_5', 'Itua':'team_2', 'Olabisi':'team_4'}
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malik</td>
<td>team_5</td>
</tr>
<tr>
<td>Itua</td>
<td>team_2</td>
</tr>
<tr>
<td>Olabisi</td>
<td>team_4</td>
</tr>
</tbody>
</table>
Dictionaries: Modification

• Modification
  – Change Malik’s team:
    \[
    \text{student}['\text{Malik}'] = '\text{team}_6'
    \]
Dictionaries: Modification

• Modification:
  – Add a new student:

```python
student_team[‘Sanusi’] = ‘team_4’
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malik</td>
<td>team_6</td>
</tr>
<tr>
<td>Itua</td>
<td>team_2</td>
</tr>
<tr>
<td>Olabisi</td>
<td>team_4</td>
</tr>
<tr>
<td>Sanusi</td>
<td>Team_4</td>
</tr>
</tbody>
</table>
Dictionaries

• Suppose someone gives you a list of students, student_list

• How can we use our dictionary, student_team, to print out the teams of each student on the student_list?
Dictionaries

• We may not know that `student_team` has an entry for an item in `student_list`.

• `def check_list(student_list):
   student_team = {"Itua":"team 2",
                   "Sanusi":"team 4"}
   for item in student_list:
      if item in student_team:
         print student_team[item]
      else:
         print 'unknown team'

• Later on: exception handling.
Useful Questions to determine what data structure to use

• Is the data I’m storing going to change?
  – Mutability VS Immutability
  – If Immutable ➔ Use a **Tuples**!

• If data will change? Can it fit into a single list?
  – If YES ➔ Use a **List**!
  – Recall a List has: *add, remove* and *sort* methods
Useful Questions

• Will one set of data be mapped to another?
  – Words to definitions, soccer students to jersey sizes, students to grades
  – If Yes: Use a *Dictionary!*
Control Structure (Decisions)
Control Flow
Do it Once with If...Elif...Else

def printFirstLetter(words):
    if words[0].isalpha():
        print(words)
    elif words[0].isspace():
        print("space")
    else:
        pass

>>>printFirstLetter("Hello")
Hello

>>>printFirstLetter(" ")
space

>>>printFirstLetter("555")
Control Flow: Loops

• What if you wanted to parse through a sentence and output each word.
  – Like “How are y555ou” to get
    » How are you

• A loop would be the best way to go about it. That way, you don’t have to write code of every alphabet in the sentence.

Let’s see an example!
def parsewords_while(words):
    currentword = ""
    index = 0
    while index < len(words):
        if words[index].isalpha():
            currentword += words[index]
        elif words[index].isspace():
            print currentword,
            currentword = ""
        else:
            pass
        index += 1
    else:
        pass

>>>parsewords_while("How are y555ou ")
How are you
For Loop
For...Else

def parsewords_for(words):
    currentword = ""
    for index in range(len(words)):
        if words[index].isalpha():
            currentword += words[index]
        elif words[index].isspace():
            print currentword,
            currentword = ""
        else:
            pass
    else:
        print currentword

>>> parsewords_for("How are y555ou ")
How are you

>>> range(5)
[0, 1, 2, 3, 4]
>>> range(2,6)
[2, 3, 4, 5]
>>> range(-10,5,2)
[-10, -8, -6, -4, -2, 0, 2, 4]
"for" Loop: Different iterations

For...Else (Alternate)

```python
def parsewords_for(words):
    currentword = ""
    for char in words:
        if char.isalpha():
            currentword += char
        elif char.isspace():
            print currentword,
            currentword = ""
        else:
            pass
    else:
        print currentword

>>> parsewords_for("How are y555ou ")
How are you
```
Nested Loops – Loop(s) within a loop

```python
def parsewords_for(words):
    currentword = ""
    for char in words:
        if char.isalpha():
            currentword += char
        elif char.isspace():
            for letter in currentword:
                print(letter,
            currentword = ""
        else:
            print(currentword)

>>>parsewords_for("How are y555ou ")
How are you
```
Summary
Why we need control structures:

• Decide what to do next
• Do certain actions for certain events
• Repeat a series of actions
• Break a series of actions
Functions
Weather forecast in Ibadan

Saturday  35°C
Sunday    30°C
Monday    38°C
Tuesday   45°C

\[ f = \frac{9}{5} c + 32 \]

Temperature in Fahrenheit

Temperature in Celsius
Converting Celsius to Fahrenheit

\[ f = \frac{9}{5}c + 32 \]

```
tempC = 21
tempF = ((9.0 / 5.0) * tempC) + 32.0
print 'Saturday:', tempF, 'F'
```

```
Saturday: 69.80000000000001 F
```

But we want the whole forecast, not just one day

```
temp_sat_C = 21  # Saturday's forecast in C
temp_sun_C = 19  # Sunday's forecast in C
temp_mon_C = 23  # Monday's forecast in C
temp_tues_C = 26  # Tuesday's forecast in C...
```
Converting Celsius to Fahrenheit

What if we want to spell out 'Fahrenheit' instead of 'F'?
Must change everywhere!

```python
# Saturday's forecast in C
temp_sat_C = 21
# Sunday's forecast in C
temp_sun_C = 19
# Monday's forecast in C
temp_mon_C = 23

# What if we want to spell out 'Fahrenheit' instead of 'F'?
# Must change everywhere!

temp_sat_F = (9.0 / 5.0) * temp_sat_C + 32.0
print 'Saturday:', temp_sat_F, 'F'
temp_sun_F = (9.0 / 5.0) * temp_sun_C + 32.0
print 'Sunday:', temp_sun_F, 'F'
temp_mon_F = (9.9 / 5.0) * temp_mon_C + 33.0
print 'Monday:', temp_mon_F, 'F'
```
Functions
Defining a function

function name, follows same naming rules as variables

name for each parameter

function body

```python
def print_as_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    print f, 'F'
```
def print_as_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
Flow of execution

```
def print_as_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
```

Program execution always starts at the first line that is *not* a statement inside a function.
def print_as_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)

Flow of execution

Function calls are like detours in the execution flow.
Flow of execution

```python
def print_as_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
```
Flow of execution

def print_as_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)

69.80000000000001 F
Flow of execution

```python
def print_as_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
```

69.80000000000001 F
def print_as_fahrenheit(c, day):
    f = ((9.0 / 5.0) * c) + 32.0
    print day + ':', f, 'F

print_as_fahrenheit(21, 'Saturday')

Saturday: 69.80000000000001 F

def print_forecast_intro():
    print 'Welcome to your weather forecast!'

print_forecast_intro()
```python
def print_as_fahrenheit(c, day):
    f = ((9.0 / 5.0) * c) + 32.0
    print day + ' :', f, 'F'
```

- What happens here?

```python
print_as_fahrenheit(21)
```

TypeError: print_as_fahrenheit() takes exactly 2 arguments (1 given)

```python
print_as_fahrenheit(21, 'Saturday', 'Sunday')
```

TypeError: print_as_fahrenheit() takes exactly 2 arguments (3 given)

```python
print_as_fahrenheit('Saturday', 21)
```

TypeError: can't multiply sequence by non-int of type 'float'
Returning a value

```python
def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f
```

A return statement ends the function immediately.

```
return EXPRESSION
```

any expression, or nothing
What is the output here?

```python
def convert_to_fahrenheit(c):
    print 'Celsius:' ,c
    f = ((9.0/5.0)*c) + 32.0
    return f
    print 'Farenheit:', f

convert_to_fahrenheit(27)
```

Celsius: 27
def absolute_value(c):
    if c < 0:
        return -c
    else:
        return c

If c is negative, the function returns here.
More than one return statement

def absolute_value(c):
    if c < 0:
        return -c
    return c

Good rule: Every path through the function must have a return statement. If you don't add one, Python will add one for you that returns nothing (the value None).
Functions can call functions

def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f

def print_as_fahrenheit(c):
    f = convert_to_fahrenheit(c)
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
Functions can call functions

def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f

def print_as_fahrenheit(c):
    f = convert_to_fahrenheit(c)
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
Functions can call functions

```python
def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f

def print_as_fahrenheit(c):
    f = convert_to_fahrenheit(c)
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
```
Functions can call functions

```python
def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f

def print_as_fahrenheit(c):
    f = convert_to_fahrenheit(c)
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
```
def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f

def print_as_fahrenheit(c):
    f = convert_to_fahrenheit(c)
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
Functions can call functions

def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f

def print_as_fahrenheit(c):
    f = convert_to_fahrenheit(c)
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
Functions can call functions

def convert_to_fahrenheit(c):
    f = ((9.0 / 5.0) * c) + 32.0
    return f

def print_as_fahrenheit(c):
    f = convert_to_fahrenheit(c)
    print f, 'F'

temp_sat_C = 21
print_as_fahrenheit(temp_sat_C)
End of Lecture!