Accelerating Information Technology Innovation

http://aiti.mit.edu

Kenya Summer 2011
Lecture 3 – Control Structures, Decisions
Beyond sequential execution

So far, all our programs have looked like this:

\[
\begin{align*}
&\text{Start with first command.} \\
&\text{Execute commands in order until there are no more.}
\end{align*}
\]

But often sequential execution is not enough.

\[
\begin{align*}
&\text{If something is true, execute the first command. Otherwise, execute the second command.}
\end{align*}
\]
Control statements

• **Conditionals:** control which set of statements is executed.
  – if / else

• **Iteration:** control how many times a set of statements is executed.
  – while loops
  – for loops
The if statement

- If the condition is True, the body gets executed.
- Otherwise, nothing happens.

```python
if x < 0:
    print 'x is negative'
```

- NOTE: IDLE editor helps with indentation.
The if/else statement

- If the condition is True, body1 gets executed.
- Otherwise, body2 gets executed.

```python
if x < 0:
    print 'x is negative'
else:
    print 'x is positive or zero'
```
Chained conditionals

- If the condition1 is True, body1 gets executed.
- Otherwise, if condition2 is True, body2 gets executed.
- If neither condition is True, body3 gets executed.
An example

```
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
    print 'I also love purple.'
```

What does this output?  

I love green.
An example

```python
a = False
b = True
if a and b:
    print 'I love red.'
elif a or b:
    print 'I love green.'
else:
    print 'I love blue.'
print 'I also love purple.'
```

What does this output?

I love green.
I also love purple.
Nested conditionals

```python
if is_adult:
    if is_senior_citizen:
        print 'Admission $2 off.'
    else:
        print 'Full price.'
else:
    print 'Admission $5 off.'
```

- Can get confusing. Indentation helps to keep the code readable and the python interpreter happy!
Another example

```python
x = 4
y = -3
if x < 0:
    if y > 0:
        print x + y
    else:
        print x - y
else:
    print x * y
```

What does this output?  

\(-12\)
Common if errors

• Syntax errors
  – Mixing up = and == in the condition

```python
b = False
if b = False
    print b
print 'inside if maybe'
```

SyntaxError: invalid syntax

IndentationError: unindent does not match any outer indentation level
The while loop

• As long as the condition is true, the body gets executed repeatedly.
• The first time the condition is false, execution ends.
The while loop

i = 0
while i < 3:
    print i
    i = i + 1

• What does this output?

0
1
2

Side note: if the condition is false the first time it is tested, the body is never executed
The break statement

• Immediately exits the innermost loop.

```python
while True:
    line = raw_input('>>> ')
    if line == 'done':
        break
    print line
print 'Done!'```

(An if statement is not a loop!)
What' will happen with this code?

\[
\begin{align*}
\text{i} &= 0 \\
\text{while } i < 3: \\
&\quad \text{print } i
\end{align*}
\]

• It will loop forever (aka Infinite loop)! How do we fix it?

\[
\begin{align*}
\text{i} &= 0 \\
\text{while } i < 3: \\
&\quad \text{print } i \\
&\quad i = i + 1
\end{align*}
\]
The infinite loop

```
i = 4
while i > 0:
    print i
    i = i + 1
```

• This code also loops forever!
• Why? And how do you fix this?

```
i = 4
while i > 0:
    print i
    i = i - 1
```
Lists

• A list is a sequence of values.
• Each element (value) is identified by an index.
• The elements of the list can be of any type.
Lists

• A list is a sequence of values.
• Each element (value) is identified by an index.
• The elements of the list can be of any type.

tens = [10, 20, 30, 40]
coins = ['dime', 'nickel', 'quarter', 'penny']
empty = []
Lists

• A list is a sequence of values.
• Each element (value) is identified by an index.
• The elements of the list can be of any type.

```
tens = [10, 20, 30, 40]
cities = ['Nairobi', 'Mombasa', 'Kisumu', 'Nakuru']
empty = []
```

• Lists can have mixed types in them, even other lists (nested).

```
mixed = ['hello', 2.0, 5, [10, 20]]
```
Creating a list

• Use the [] brackets

```
list_of_ints = [10, 20, 30, 50]
```

Only one name

Four int values
List operators

- Applied to lists, produce lists

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenation</td>
</tr>
<tr>
<td>*</td>
<td>Repetition</td>
</tr>
<tr>
<td><code>&lt;list&gt;[ ]</code></td>
<td>Indexing</td>
</tr>
<tr>
<td><code>&lt;list&gt;[ : ]</code></td>
<td>Slicing</td>
</tr>
</tbody>
</table>
Accessing list elements

• Individual elements are accessed using the [] operator.

```
list_of_ints[0] = 17
```

Lists are mutable!
Assigns the first element to 17

```
list_of_ints = [17, 20, 30, 50]
```

List indexing starts at 0, not 1!

```
new_var = list_of_ints[0]
```

accesses the value of the first element

```
new_var = 17
```

now also has value 17
Printing a list

- We can use the print function to output the contents of the list:

```python
cities = ['Nairobi', 'Mombasa', 'Kisumu']
numbers = [17, 123]
empty = []
print cities, numbers, empty
```

`['Nairobi', 'Mombasa', 'Kisumu'] [17, 123] []`
An example

```python
numbers = [10, 20, 30]
letters = ['a', 'b', 'c']
number = numbers[0]
letter = letters[2]
print 'number =', number
print 'letter:', letter
print 'letters:', letters
```

```
number = 10
letter: 'c'
letters: ['a', 'b', 'c']
```
Another example

```python
numbers = [10, 20, 30]
letters = ['a', 'b', 'c']
mixed = letters + numbers
print mixed
print letters*2
numbers[2] = letters
print numbers
print numbers[:2]
numbers[1:] = [40, 50]
print numbers
```

```
['a', 'b', 'c', 10, 20, 30]
['a', 'b', 'c', 'a', 'b', 'c']
[10, 20, ['a', 'b', 'c']]
[10, 20]
[10, 40, 50]
```
Out-of-range errors

• You will get a runtime error if you try to access an element that does not exist!

```
list_of_ints = [17, 9, 42, -2]
print list_of_ints[4]
```

```
list_of_ints  17  42 -3  9
```

```
list_of_ints[4] doesn’t exist!
```

`IndexError: list index out of range`
Lists vs. Strings

• Lists are mutable - their contents can be modified
• Strings are immutable

```python
name = 'Lenny'
name[0] = 'J'
```

`TypeError: object doesn't support item assignment`
The for loop

For loop syntax:

```
for ELEMENT in SEQUENCE:
    BODY
```

- **ELEMENT** is the sequence element.
- **SEQUENCE** is the sequence of values (list, string, etc.).
- **BODY** is any set of statements.
- Indentation is important.

**Example:**

```
for i in [0,1,2,3]:
    print i
```

Output:

```
0
1
2
3
```
Using range

index variable

for INDEX in range(n):
    BODY

any set of statements

generates sequence of n values starting at 0 and incrementing by 1

• What does this output?

for i in range(4):
    sq = i * i
    print i, sq

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
Using range

index variable

for INDEX in range([start], stop, [step]):
  BODY

generates sequence of values
start and step are optional

any set of statements

• What does this output?

```python
for i in range(1, 7, 2):
    print i
```

1
3
5
For loop and strings

• Iterating through the characters of a string

```python
str1 = 'stressed'
for c in str1:
    print c,
```
For loop and strings

- What is the output?

```python
str1 = 'stressed'
res = ''
for c in str1:
    res = c + res
print res
```

desserts
For loop and lists

• Iterating through the elements of a list

desserts = [['ice cream', 3.5], ['chocolate cake', 2], ['pudding', 3]]

for dessert_list_item in desserts:
    print dessert_list_item[0], 'costs $', dessert_list_item[1]

ice cream costs $ 3.5
chocolate cake costs $ 2
pudding costs $ 3
Exercise

• Is there another way we can get the same output from the ‘desserts’ list?

desserts = [['ice cream', 3.5], ['chocolate cake', 2], ['pudding', 3]]

for dessert_list_item in desserts:
    print dessert_list_item[0], 'costs $', dessert_list_item[1]

    ice cream costs $ 3.5
    chocolate cake costs $ 2
    pudding costs $ 3
Combining for and if

```python
for i in range(6):
    if i % 2 == 0:
        print i, 'is even.'
    else:
        print i, 'is odd.'
```

• What does this output?

0 is even.
1 is odd.
2 is even.
3 is odd.
4 is even.
5 is odd.
Nested for loops

for i in range(1, 6):
    for j in range(1, 6):
        prod = i * j
        # use comma to print all on one line
        print prod,
    print

• What does this output?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
For vs While

• For loop is primarily used
  • for iterating over a sequence of values
  • when we know the number of iterations in advance

• While loop is primarily used
  • when we don't know the number of iterations in advance (they could be controlled by user input)
Questions?
def printFirstLetter(words):
    if words[0].isalpha():
        print words
    elif words[0].isspace():
        print "space"
    else:
        pass

>>> printFirstLetter("Hello")
Hello

>>> printFirstLetter(" ")
space

>>> printFirstLetter("555")
Loops: Break vs Continue

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 = ""
        index += 1
    print currentword

>>> parsewords_while("How are y555ou")
How are you
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 = ""
            index += 1
    print currentword

>>> parsewords_while("How are y555ou")
How are you
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 = ""
    print currentword
Loops

For (Original)

```python
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:
        print currentword

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

For (Alternate)

```python
def parsewords_for(words):
currentword = ""

for char in words:
    if char.isalpha():
        currentword += char
    elif char.isspace():
        print currentword,
        currentword = ""
    else:
        print currentword

>>> parsewords_for("How are y555ou")
How are you
```
Nested Loops

```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 = ""
    print currentword

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