Lecture 8: Conditionals & Control Flow (Sections 5.1-5.7)

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Fan, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Optional 1-on-1 with a staff member to help *just you* with course material. Sign up for a slot on CMS under “SPECIAL: one-on-ones“.

A1 first submission due Feb 19 Wedn at 11:59pm
Review: Objects are referenced

- Must **call constructor** function to **create** object
  - Object variable stores **ID of** object
- Multiple variables can reference same object
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t
swap_x(p, q)

What is in p.x at the end of this code?
A: 1
B: 2
C: 3
D: I don’t know

Heap Space

Global Space

<table>
<thead>
<tr>
<th>id1</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id2</th>
<th>Point3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
</tr>
</tbody>
</table>
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t
swap_x(p, q)

What is in p.x at the end of this code?
A: 1
B: 2
C: 3  CORRECT
D: I don’t know

Heap Space

Global Space

p  id1
q  id2
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
def swap_x(p, q):
    t = p.x
    p.x = q.x
    q.x = t
swap_x(p, q)
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)
def swap(p, q):
    t = p
    p = q
    q = t
swap(p, q)

What is in global p after calling swap?

A: id1
B: id2
C: I don’t know
import shapes
p = shapes.Point3(1, 2, 3)
q = shapes.Point3(3, 4, 5)
def swap(p, q):
    t = p
    p = q
    q = t
swap(p, q)

What is in global p after calling swap?

A: id1  CORRECT
B: id2
C: I don’t know

Global Space

p  id1
q  id2

Heap Space

id1  Point3
    x  1
    y  2
    z  3

id2  Point3
    x  3
    y  4
    z  5
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

def swap(p, q):
    t = p
    p = q
    q = t
swap(p, q)
Methods: Functions Tied to Classes

- **Method**: function tied to object
  - Method call looks like a function call preceded by a variable name:
    \[
    \langle \text{variable} \rangle \cdot \langle \text{method} \rangle(\langle \text{arguments} \rangle)
    \]

**Example:**

```python
import shapes
u = shapes.Point3(4,2,3)
u.greet()
```

"Hi! I am a 3-dimensional point located at (4,2,3)"

Where else have you seen this??
Example: String Methods

- **s₁.upper()**
  - Returns a copy of \( s_1 \) with white-space removed at ends

- **s.strip()**
  - Returns a copy of \( s \) with white-space removed at ends

- **s₁.index(s₂)**
  - Returns position of the first instance of \( s_2 \) in \( s_1 \)
  - **error** if \( s_2 \) is not in \( s_1 \)

- **s₁.count(s₂)**
  - Returns number of times \( s_2 \) appears inside of \( s_1 \)
# Built-in Types vs. Classes

<table>
<thead>
<tr>
<th><strong>Built-in types</strong></th>
<th><strong>Classes</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Built-into Python</td>
<td>Provided by modules</td>
</tr>
<tr>
<td>Refer to instances as <em>values</em></td>
<td>Refer to instances as <em>objects</em></td>
</tr>
<tr>
<td>Instantiate with <em>literals</em></td>
<td>Instantiate w/ <em>constructors</em></td>
</tr>
<tr>
<td>Can ignore the folders</td>
<td>Must represent with folders</td>
</tr>
</tbody>
</table>

So far only about understanding *objects*; later will create your own *classes*
Statements either affect **data** or **control**

- **DATA**: change the value of a variable, create a variable, *etc.*
  
  Examples:
  
  \[
  x = x + 1 \\
  name = "Alex"
  \]

- **CONTROL**: tell python what line to execute next
  
  Examples:
  
  \[
  greet(name) \\
  if name == "Alex": \Rightarrow \text{today’s Lecture}
  \]
## Conditionals: If-Statements

### Format

<table>
<thead>
<tr>
<th>if <code>&lt;boolean-expression&gt;</code>:</th>
<th><code>&lt;statement&gt;</code>… <code>&lt;statement&gt;</code></th>
</tr>
</thead>
</table>

### Example

```
# is there a new high score?
if curr_score > high_score:
    high_score = curr_score
    print("New high score!")
```

### Execution:

- If `<boolean-expression>` is true, then execute all of the statements indented directly underneath (until first non-indented statement).
What are Boolean expressions?

Expressions that evaluate to a Boolean value.

is_student = True
is_senior = False
num_credits = 25

Boolean operations:
if is_student and is_senior:
    print(“Hi senior student!”)

Boolean variables:
if is_student:
    print(“Hi student!”)

Comparison operations:
if num_credits > 24:
    print(“Are you serious?”)
What gets printed, Round 1

\[
\begin{align*}
\text{a} &= 0 \\
\text{print(a)} \\
\text{a} &= \text{a + 1} \\
\text{print(a)} \\
\text{a} &= \text{a + 1} \\
\text{print(a)} \\
\text{a} &= \text{a + 1} \\
\text{print(a)} \\
\text{a} &= \text{a + 1} \\
\text{print(a)} \\
\text{a} &= \text{a + 1} \\
\text{print(a)} \\
\end{align*}
\]
What gets printed? (Question)

```python
a = 0
if a == 0:
    a = a + 1
if a == 0:
    a = a + 2
a = a + 1
print(a)
```

A: 0
B: 1
C: 2
D: 3
E: I do not know
What gets printed? (Solution)

```python
a = 0
if a == 0:
    a = a + 1
if a == 0:
    a = a + 2
a = a + 1
print(a)
```

A: 0
B: 1
C: 2 CORRECT
D: 3
E: I do not know
## Conditionals: If-Else-Statements

### Format

<table>
<thead>
<tr>
<th>if &lt;boolean-expression&gt;:</th>
<th># new record?</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;statement&gt;</td>
<td>if curr_score &gt; high_score:</td>
</tr>
<tr>
<td></td>
<td>print(“New record!”)</td>
</tr>
<tr>
<td>else:</td>
<td>else:</td>
</tr>
<tr>
<td>&lt;statement&gt;</td>
<td>print(“Try again next time”)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### Example

```python
# new record?
if curr_score > high_score:
    print(“New record!”)
else:
    print(“Try again next time”)```

### Execution:

If `<boolean-expression>` is true, then execute statements indented under `if`; otherwise, execute the statements indented under `else`.
Conditionals: “Control Flow” Statements

if b:
    s1  # statement
    s3  # statement

if b:
    s1
    else:
        s2
        s3

Flow
Program only takes one path during an execution (something will not be executed!)
What gets printed, Round 2

a = 0
if a == 0:
    a = a + 1
else:
    a = a + 2
print(a)

a = 0
if a == 1:
    a = a + 1
else:
    a = a + 2
print(a)

a = 0
if a == 1:
    a = a + 1
else:
    a = a + 1
    a = a + 1
print(a)

a = 0
if a == 1:
    a = a + 1
else:
    a = a + 1
a = a + 1
print(a)
if determines which statement is executed next

```python
def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = True
    get_in_car(car_locked)
```
Program Flow (car locked, 1)

if determines which statement is executed next

```python
def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = True
    get_in_car(car_locked)
```

Global Space

| car_locked | True |

25
if determines which statement is executed next

def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
        car_locked = True
    get_in_car(car_locked)
Program Flow (car locked, 3)

if determines which statement is executed next

def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = True
get_in_car(car_locked)
### Program Flow (car locked, 4)

**if** determines which statement is executed next

```python
def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = True
get_in_car(car_locked)
```

**Global Space**
- car_locked: True

**Call Frame**
- get_in_car: 1/3
- car_locked: True

Unlock car!
if determines which statement is executed next

def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = True
    get_in_car(car_locked)

Unlock car!
Open the door.
Program Flow (car not locked, 0)

if determines which statement is executed next

```python
def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = False
get_in_car(car_locked)
```

Global Space
Program Flow (car not locked, 1)

if determines which statement is executed next

def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = False
    get_in_car(car_locked)
Program Flow (car not locked, 2)

**if** determines which statement is executed next.

```python
def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
    print("Open the door.")
    car_locked = False
get_in_car(car_locked)
```

**Global Space**
- `car_locked`: False

**Call Frame**
- `get_in_car`
- `car_locked`: False

---

32
Program Flow (car not locked, 3)

if determines which statement is executed next

def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
    print("Open the door.")
    car_locked = False
get_in_car(car_locked)
### Program Flow (car not locked, 4)

**if** determines which statement is executed next

```python
def get_in_car(car_locked):
    if car_locked:
        print("Unlock car!")
        print("Open the door.")
    car_locked = False
    get_in_car(car_locked)
```

- **Global Space**
  - `car_locked`: False

- **Call Frame**
  - `get_in_car`
  - `car_locked`: False
  - RETURN: None

Open the door.
def max(x, y):
    if x > y:
        return x
    return y

max(0, 3)

Current call frame:

<table>
<thead>
<tr>
<th>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>
What does the call frame look like next? (Q)

def max(x,y):
    if x > y:
        return x
    return y

max(0,3)

Current call frame:
```python
def max(x, y):
    if x > y:
        return x
    return y
```

max(0, 3):

```
Table:
<table>
<thead>
<tr>
<th>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>
```
Call Frame Explanation (2)

```python
def max(x, y):
    if x > y:
        return x
    return y
```

```
max(0, 3):
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>1 3</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

Skips line 2
Call Frame Explanation (3)

```python
def max(x, y):
    if x > y:
        return x
    return y
```

`max(0, 3):`

```
<table>
<thead>
<tr>
<th>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>
```

RETURN 3
Variables created inside **if** continue to exist past **if**: 

```python
a = 0
if a == 0:
    b = a + 1
print(b)
```

...but are only created if the program actually executes that line of code
def max(x,y):
    """Returns: max of x, y"""
    # note: code has a bug!
    # check if x is larger
    if x > y:
        bigger = x
    return bigger

maximum = max(3,0)

Value of maximum?

A: 3
B: 0
C: Error!
D: I do not know
def max(x,y):
    """Returns: max of x, y"""
    # note: code has a bug!
    # check if x is larger
    if x > y:
        bigger = x
    return bigger

maximum = max(3,0)

Value of maximum?

A: 3  CORRECT
B: 0
C: Error!
D: I do not know

- Local variables last until
  - They are deleted or
  - End of the function
- Even if defined inside if
Control Flow and Variables (Q2)

def max(x,y):
    """Returns: max of x, y"""
    # note: code has a bug!
    # check if x is larger
    if x > y:
        bigger = x
    return bigger

maximum = max(0,3)

Value of maximum?

A: 3
B: 0
C: Error!
D: I do not know
def max(x, y):
    """Returns: max of x, y""
    # note: code has a bug!
    # check if x is larger
    if x > y:
        bigger = x
    return bigger

maximum = max(0, 3)

Value of maximum?

A: 3
B: 0
C: Error! CORRECT
D: I do not know

- Variable existence depends on flow
- Generally terrible idea to refer to variables defined inside an if clause
def zero_or_one(a):
    if a == 1:
        b = 1
    else:
        b = 0
    print(b)

make sure that ALL if branches create the variable
Conditionals: If-Elif-Else-Statements

**Format**

```python
if <Boolean expression>:
    <statement>
    ...
elif <Boolean expression>:
    <statement>
    ...
...
else:
    <statement>
    ...
```

**Example**

```python
# Find the winner
if score1 > score2:
    winner = “Player 1”
elif score2 > score1:
    winner = “Player 2”
else:
    winner = “Players 1 and 2”
```
Conditionals: If-Elif-Else-Statements

Format

if <Boolean expression>:
    <statement>
    ...
elif <Boolean expression>:
    <statement>
    ...
else:
    <statement>
    ...

Notes on Use

- No limit on number of elif
  - Must be between if, else
- else is optional
  - if-elif by itself is fine
- Booleans checked in order
  - Once Python finds a true <Boolean-expression>, skips over all the others
  - else means all <Boolean-expression> are false
If-Elif-Else (Question)

```python
a = 2

if a == 2:
    a = 3
elif a == 3:
    a = 4

print(a)
```

What gets printed?

A: 2  
B: 3  
C: 4  
D: I do not know
If-Elif-Else (Answer)

```python
a = 2

if a == 2:
    a = 3
elif a == 3:
    a = 4
print(a)
```

What gets printed?

- A: 2
- B: 3  **CORRECT**
- C: 4
- D: I do not know
What gets printed, Round 3

```python
a = 2
if a == 2:
a = 3
elif a == 3:
a = 4
print(a)
```

```
a = 2
if a == 2:
a = 3
if a == 3:
a = 4
print(a)
```

3

4
def what_to_wear(raining, freezing):
    if raining:
        if freezing:
            print("Wear a waterproof coat.")
        else:
            print("Bring an umbrella.")
    else:
        if freezing:
            print("Wear a warm coat!")
        else:
            print("A sweater will suffice.")
Can use print statements to examine program flow

'before if'
'inside if x>y'
'after if'

x must have been greater than y

# Put max of x, y in z
print('before if')
if x > y:
    print('inside if x>y')
    z = x
else:
    print('inside else (x<=y)')
    z = y
print('after if')

"traces" or "breadcrumbs"
x must have been greater than y