Decision Making

George Mason University
Today’s topics

- Review of Chapter 2: Decision Making
- Go over exercises
- Decision making in Python
What is a decision statement?

1. `age = get the age from the user`
2. `if the age > 20:
   True:
      `result = “Congrats! You can now rent the Two Door Speck!”`
   False:
      `result = “Enjoy your bicycle, uphill both ways in the snow.”`
3. `return result`

• Must always evaluate to true or false
• The decision must be composed of variables we have already given values to
Two types of decisions

- Nested decisions remember the results of decisions made before them (in the same nesting)
- Independent decisions do not
Boolean values and calculations

- A boolean value must evaluate to true or false
- Two boolean values can be compared with **and** or **or**
- Use parentheses if you want to combine **and** and **or**, i.e. \((x \text{ and } y) \text{ or } z\) or \(x \text{ and } (y \text{ or } z)\), to disambiguate

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>x and y</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>x or y</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Calculations that evaluate to boolean values

- $< \leq > \geq$ all evaluate to true or false
  
  $3 < 2$ is false

- is, is not also evaluate to true or false
  
  $3$ is $3$ is true

  “jello” is not “blue” is true
Let’s go over examples
if statements in Python

5. if operation is addition:
   True: result = number1 + number2
   False: do nothing
6. if operation is subtraction:
   True: result = number1 - number2
   False: do nothing
7. if operation is multiplication:
   True: result = number1 * number2
   False: do nothing
8. if operation is division:
   True: result = number1 / number2
   False: x = 3

   if operation is "addition":
       result = number1 + number2
   if operation is "subtraction":
       result = number1 - number2
   if operation is "multiplication":
       result = number1 * number2
   if operation is "division":
       result = number1 / number2

• **if** is a keyword; the if statement must end in a colon
• **is** is a keyword that compares the variable operation to a string
• what belongs to a particular if statement is indented
elif statements in Python

- **elif** is a keyword; it stands for else if
- **elif** is attached to an `if` or *elif* before it, and indicates this *elif* is nested
- (you cannot have a standalone *elif*)
if versus elif

The one on the left returns 3
The one on the right returns 6
Why? if-elif statements are nested, linked, and mutually exclusive. The plain if statements are not mutually exclusive, don’t know about each other, and all get executed
else statements in Python

1. `number1` = get the first number from the user
2. `number2` = get the second number from the user
3. if `((number1 - number2) is 1) or ((number1 - number2) is -1):
   True: `result = "consecutive"
   False: `result = "not consecutive"
4. `return result`

```python
num1 = input("Enter number1: ")
num2 = input("Enter number2: ")
if `((num1 - num2) is 1) or ((num1 - num2) is -1):
    result = "consecutive"
else:
    result = "not consecutive"
return result
```

- **else** is a keyword; it is linked to an if or elif, and gets executed if the if/elif above it is false
else statements in Python

```python
operation = "addition"
if operation is "addition":
    result = 3
elif operation is "addition":
    result = 4
elif operation is "addition":
    result = 5
elif operation is "addition":
    result = 6
else:
    result = "operation was not addition"
return result
```

- **else** only gets executed if none of the **if** or **elif** before it are true
Indentation matters!

```python
result = ""
if num1 is 0:
    result = result + "num1 is "
    result = result + " 0"
elif num1 is 1:
    result = result + "num1 is 1"
    if num2 > 3:
        result = result + "num2 is > 3"
    elif num2 > 4:
        result = result + "this will NEVER run"
    else:
        result = result + "num2 is <= 3"
        result = result + "finished num1 is 1"
else:
    result = result + "num1 is not 0 or 1"
```

This is another type of “nesting”, and what people usually refer to as nested if-else statements
Why do we have two ways to compare for equality?

\[ y = 5 \]

\[ y \text{ is } 5 \] returns true
\[ y \text{ is } 5.0 \] returns false

\[ y == 5 \] returns true
\[ y == 5.0 \] returns true

It is **ALWAYS SAFER** to use `==` instead of `is`, even when comparing things like strings for equality in Python. `is` compares memory addresses of variables, which gets confusing with strings.
Assignment Statement \( x = 5 \)

Boolean Expression \( x == 5 \)
boolean types in Python

```python
value1 = (1 == 1)
value2 = True
value3 = False
return (value1 and value2 and not value3)
```

• **True** and **False** are both keywords and types in Python (capitalization matters)

• **not** is a keyword that negates a boolean value

• the code above returns True
Questions?