What is a Decision Statement?

• A statement that evaluates some conditions to **true** or **false**.

```python
if condition:
    statement
    statement ...
```

• A condition (or expression)
  • Must always evaluate to true or false, i.e., “Boolean expression”
  • Use relational operators: >, <, >=, <=, ==, !=

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

age 19
Calculations that Evaluate to Boolean Values

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

• $==$, $!=$ also evaluate to true or false
  3 $==$ 3 is true
  3 $==$ 4 is false
  “jello” $!=$ “blue” is true

age $>$ 21

“Boolean value”
Decision Structure

Simple structure

```plaintext
if condition:
    statements1
else:
    statements2
```

Dual Structure

```plaintext
if condition:
    statements1
else:
    statements2
```

Indent the block of statement
Two Types of Decisions

**Sequence Decision**

```python
if condition:
    block of statements ...
if condition:
    block of statements ...
if condition:
    block of statements ...
else:
    block of statements ...
```

**Nested Decision**

```python
if condition:
    block of statements ...
eelif:
    block of statements ...
eelif:
    block of statements ...
else:
    block of statements ...
```

- Nested decisions remember the results of decisions made before them (in the same nesting)
- Independent decisions do not
if Statements in Python

• **if** is a keyword, the if statement must end in a colon
• What belongs to a particular if statement is **indented**
elif Statements in Python

```python
if operation is addition:
    True: result = number1 + number2
    False: if operation is subtraction:
        True: result = number1 - number2
        False: if operation is multiplication:
            True: result = number1 x number2
            False: x = 3
elif operation == "subtraction":
    result = number1 - number2
elif operation == "multiplication":
    result = number1 * number2
elif operation == "division":
    result = number1 / number2
```

- **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 6
- if-elif statements are nested, linked, and mutually exclusive.

- The one on the right returns 4
- The plain if statements are not mutually exclusive, don’t know about each other, and thus all if statements get executed.
else statements

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, linked to an `if` or `elif`, and get executed if the `if/elif` above it is false
**else statements (2)**

```python
if operation == "addition":  
    result = number1 + number2

elif operation == "subtraction":
    result = number1 - number2

elif operation == "multiplication":
    result = number1 * number2

elif operation == "division":
    result = number1 / number2
else:
    result = "operation undefined"
```

- **else** only gets executed if none of the **if** or **elif** before it are true.
This is another type of “nesting”, and is usually referred to as “nested if-else statements”
Indentation Matters

```python
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
        else:
            result += "num2 <= 3"
    else:
        result += "num1 is not 0 or 1"
    return result

print(template(0, 1))  # "num1 is 0"
print(template(1, 3))  # "num1 is 1 num2 <= 3 finished num1"
print(template(1, 2))  
print(template(2, 1))  
```
**Indentation Matters**

```python
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
    else:
        result += "num2 <= 3"
        result += " finished num1"
    else:
        result += "num1 is not 0 or 1"
    return result

print(template(0, 1))  # "num1 is 0"
print(template(1, 3))  # "num1 is 1 num2 <= 3 finished num1"
print(template(1, 2))  # "num1 is 1 num2 <= 3 finished num1"
print(template(2, 1))  # "num1 is not 0 or 1"
```
Indentation Matters

```python
def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result += "num2 > 3"
        elif num2 > 4:
            result += "THIS WILL NEVER RUN"
    else:
        result += "num2 <= 3"
        result += " finished num1"
    else:
        result += "num1 is not 0 or 1"
    return result
```

Indentation groups things

- if, elif, and else are mutually exclusive

print(template(0, 1))  # "num1 is 0"
print(template(1, 3))  # "num1 is 1 num2 <= 3 finished num1"
print(template(1, 2))  # "num1 is 1 num2 <= 3 finished num1"
print(template(2, 1))  # "num1 is not 0 or 1"
Unreachable statements

def template(num1, num2):
    result = ""
    if num1 == 0:
        result = "num1 or 0"
    elif num1 == 1:
        result = "num1 is 1"
        if num2 > 3:
            result += "num2 > 3"
    elif num2 > 4:
        result += "THIS WILL NEVER RUN"
    else:
        result += "num2 <= 3"
        result += " finished num1"
    return result

print template(1, 5)
Programming TRAP

• Assignment statement  \( x = “CS1111” \)

• Boolean expression  \( x == “CS1111” \)
Boolean Types

```python
def boolean_example():
    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 !!

- **not** is a keyword that negates a Boolean value

- The code above returns **True**
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` or `or` to disambiguate; e.g., 
  \[ (x \text{ and } y) \text{ or } z \] or \[ x \text{ and } (y \text{ or } z) \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</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>x</th>
<th>y</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>

- You can use any logical operators: `and` or `or` or `not`