Chapter 6 - If Statements

Goals:

- Correctly write programs that use if and else statements
- Use Boolean expressions in if statements
- Trace the execution of if statements both inside and outside of loops

6.0 Use `if` statements to control whether a block of code is executed

- An `if` statement (more properly called a *conditional* statement) controls whether some block of code is executed or not
- The first line starts with `if` and ends with a colon
- The body containing one or more statements is indented (usually 4 spaces)
- In the following code cell, we prompt the user to enter a temperature (in F), test if the temperature is greater than 90 F, and, if so, print a statement.
- Run the following code cell a couple times to try out different input temperatures

```
# prompt user to enter the temperature in F
temp = int(input("Enter a temperature in F:  "))

# See if the temp is greater than 90 F or not
if temp > 90:
    print(temp,'is too hot!!')
```

- You should find that if the temperature is > 90 F, a message is printed. Otherwise it isn't
- BTW, notice that we placed the `input()` statement inside a `int()` function to convert the returned string into an integer that we can use with the > operator.

6.2 Use `else` to execute a block of code when an `if` condition is not true

- `if` statements can stand alone (as in the first example), or they can be combined with an `else` statement.
- The `else` statement allows us to run an alternative code block when the `if` condition is false.

Let's extend our code above to make it more general. We'll add a print statement if the if condition fails (i.e. if the temperature is less than or equal to 90 F). Again, try running the code for a few input temperatures and see if the output makes sense.

```
# prompt user to enter the temperature in F
temp = int(input("Enter a temperature in F:  "))

# See if the temp is greater than 90 F or not
if temp > 90:
    print(temp,'is too hot!!')
else:
    print(temp,'is not too hot')
```

6.3 Use `elif` to specify additional tests

- `elif` is short for "else if"
- The `elif` statement must come after the `if` statement and before and `else` statement (if one is used)
- When combined with `elif` statements, the `else` statementis a "catch all" and is executed if none of the `if` or `elif` statements are true

🔆 Example: Temperature classification

- The following example prompts the user to enter a temperat
- The code classifies what kind of day it is based on the temp.

- Run it a few times with different temperatures to see how it behaves

```
temp = int(input("Enter a temperature in F:  "))        # temperature in F

# Classify temperature. Temp divisions:  32, 60, 90
if temp > 90:
    print(temp,'is too hot!!  Spend the day at the beach.')
elif temp > 60:
    print(temp,'is nice. Go outside and play.')
elif temp > 32:
    print(temp,'is chilly. Wear a jacket')
else:
    print(temp,'is too cold!!  Stay inside and code Python.')
```

✅ Skill Check 1

- The order of the `if`, `elif` and `else` statements matters. The following code scrambles the order of the conditional checks.
- Run the program and try different temperatures.
- Create a text box and describe how the code behaves. The following code prompts the user for a temperature. Explain why this version of the code is broken.

```
temp = int(input("Enter a temperature in F:  "))        # temperature in F

if temp > 40:      # if temperature is > 40, print the following
    print(temp,'is chilly')
elif temp > 70:    # else if temperature is > 70, print the following
    print(temp,'is nice')
elif temp > 90:    # else if temperature is > 90, print the following
    print(temp,'is too hot!!')
else:              # otherwise, print the following
    print(temp,'is too cold!!')
```

🟩

6.5 Comparison (i.e. relational) and Boolean operators

In addition to > and <, Python offers the following comparison (i.e. relational) operators:

```
#    ==   Equality operator          x == y     True if x equal y
#    !=   Not equal                  x != y     True if x is not equal to y
#    >    Greater than               x > y      True if x is greater than y
#    <    Less than                  x < y      True if x is less than y
#    >=   Greater than or equal to   x >= y     True if x is greater than or equal to y
#    <=   Less than or equal to      x <= y     True if x is less than or equal to y
#    is   same object (identity)     a is b     True if a and b are the same object (not just numerically equal)
#    in   membership                 a in b     True if a is in b
```

These comparisons result in True/False states, which can be further combined with the following Booelan operators:

```
#    and    logical AND      x and y     True if BOTH x and y are True
#    or     logical OR       x or y      True if EITHER x or y are True
#    not    logical NOT      not x       True if x is False
```

Each of these comparisons results in a `True` or `False` Boolean. Run the following:

```
2 > 1
```

Try this one:

```
(1>2) or (2<3)
```

And one more:

```
not (1<2)
```

Here are some examples of using comparisons in if statements:

```
A = [0, 10, 20, 40, 100]
x = 10

if x >= A[0] and x < A[2]:
    print("first condition is true")

if x == A[1]:
    print("second condition is true")

if x in A:
    print("third condition is true")

if not(999 in A):
    print("fourth condition is true")
```

✅ Skill Check 2

Determine whether each of the following conditions produces True or False without executing the code. Create a Text Cell to type your answers.

```
4 > 3
2-1 == 3-2
(4*6 > 5*5) or 2>1
(2>1) and (3>1) and (4>1) and (4>5)
```

✅ Skill Check 3

Classify a number.

- Write a program to prompt the user to enter a number.
- Don't forget to convert the input to a float
- Print a message stating whether the number is an integer or if it has nonzero decimal places.
- If the number is an integer, print a message saying if it is even or odd (hint: you can use the % operator)

✅ Skill Check 4

The x and y coordinates for 8 points are given in the code cell below.

- Write some code to classify which quandrant each point is in.
- If a point lies exactly on the x or y axis, it doesn't belong to a quadrant. Instead, say which axis it is on.
- Your code should output nicely-formatted (x,y) coordinates of each point along with your classification. The first few lines of the output should look something like the following:

```
( 2, 8)   quadrant I
(-4, 2)   quadrant II
(-2, 0)   x axis
```

```
x = [2, -4, -2, -5, 1, -6,  3, -2]
y = [8,  2,  0,  4, 9, -3, -3,  0]
```

6.6 Coding Patterns: If statements used inside loops

- Loops, if statements and variable update rules can be combined in many, many ways.
- Some combinations come up so often they are given names. These code structures are called "**coding patterns**"
- Here are a few of the most common coding patterns we will encounter:

The Update (or Replacement) Pattern updates old information with new

- Unlike the Accumulator Pattern (discussed in the last chapter) in which new information is added to or combined with old information, the Update Pattern replaces old with new.

Example: Finding the max or min of a list

- A common application application of the Update Pattern is looking for the maximum or minimum value in a list of numbers
- In the following code, a list is defined
- The mmaximum value `max_value` is initially assigned to the first element in the list
- The rest of the list is scanned using a for loop
- If a list element is greater than the stored `max_value`, the `max_value` is assigned to that element

Run the following Code Cell to try it out.

```
vlist = [3, 7, 9, -5, 27, -2, 12]  # define a list of numbers

max_v = vlist[0]                   # initialize max_value to the first element of the list

for v in vlist:                    # loop over numbers in the list
    if v > max_v:                  # check if the current number is greater than max_val
        max_v = v                  # if the condition is true, update max_val to current number

print("maximum value = ", max_v)   # print out the max value in the list
```

A **second example** loops through a Python list to calculate the difference between consecutive pairs of elements.

- We are given a list of values `y_values`
- As we loop through the times, we want to keep track of the curent (new) value and the previous (old) value
- On each iteration, the previous "new" value becomes the current "old" value (see below)
- We calculate the difference by subtracting the old value from the new
- We'll see later, this pattern is useful for solving differential equations and other time-dependent problems

```
y_values = [0, 1, 5, 4, 2, 0]    # define a list of numbers

y_old = y_values[0]              # initialize old time to the first element
y_new = y_values[0]              # initialize new time to the first element

for y in y_values[1:]:           # loop over times, starting with 2nd element
    y_old = y_new                # current "old" time = previous "new" time
    y_new = y                    # current "new" time = current time in the loop

    diff = y_new - y_old         # Calc difference between new and old
    print("diff = ",diff)        # display difference
```

The Count Pattern counts how many times a feature is found

- In this pattern, we define a counter variable and initialize it to zero
- We loop through a list and check to see if a given condition is met. If it is, we increment the counter
- Notice we use a Python shoethand for incrementing `count`. The statement `count += 1` is shorthand for `count = count + 1`
- In the following example, we define a list of how many cats a certain group of people own.
- We count how many of the people own at least 3 cats (which is stored in a variable called threshold)

```
n_cats = [0,1,5,0,1,1,3,4,10]    # number of cats owned

count = 0                        # initialize count

threshold = 3                    # threshhold = minimum number of cats to sesarch for
for cats in n_cats:              # loop over numbers in the list
    if cats >= threshold:        # check if the current number is greater than max_val
        count += 1               # if the condition is true, update max_val to current number

print(count," people in the list own at least",threshold,"cats")
```

✅ Skill Check 5

- Ask the user to input 5 integers
- This can be done by placing an input statement inside a for loop.
- Count how many of the inputted numbers are even.
- Display the result to the screen with a message

🔆 The Search Pattern loops through a list until a feature is found

- In the following example, we determine if a given number is present in a list of numbers.
- We initialize a Boolean variable called `found` to be `False` (since the number is not found before we start looking).
- We then loop through the list and check to see if each element equals our target.
- If we find the target in our list, we set our `found` variable to `True`.
- Since we've already found the target, we don't need to keep looping, so we use the `break` command. This command exits the for loop immediately

```
numbers = [3,5,9,11,15,19,24]   # define a list of numbers
target = 15                     # target = number to search for in list

found = False                   # found = Boolean variable that equals false until a match is found
for nbr in numbers:             # loop over numbers in the list
    if nbr == target:           # check if the current number is equal to target
        found = True            # if the condition is met, set found to True
        break                   # exit the loop and stop searching

if found:
    print("numbers list contains",target)
else:
    print("numbers list does not contain",target)
```

✅ Skill Check 6

Write a program that detects peaks in a time series.

- The Code Cell below contains a series of sequential measurements `y` made at times `t`
- Your program should scan the list and find the values that are at the top of a local peak
- We define a local peak as a list element that is greater than or equal to both of its neighbors. i.e. a list element y_n is a local peak if $y_n \geq y_{n+1}$ and $y_n \geq y_{n-1}$.
- Cells at the beginning and end of the list, i.e. `y[0]` and `y[-1]` should not be considered as being local peaks since it is not possible to compare their values to both neighbors.
- The code should count the number of local peaks found
- You don't need to save the `y` and `t` values, just print them out
- Print the times and y values of the local peaks
- Print a statement saying how many local peaks were found

Before you start coding, think about which design pattern(s) are relevant to this problem. Which will you use?

Enter your answer here:

```
y = [0, 0.2, 0.5, 1.2, 2.4, 2.0, 0.4, 0.1, 0.2, 0.7, 1.3, 0.9, 0.2]   # y measurements
t = [0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12]   # times
```

✅ Skill Check 7

Write a guessing game program.

- Generate a random integer between 1 and 100. This will be the target number the user tries to guess. (see the Rolling Dice example in Chapter 5 for an example of how to use the random library to produce a random integer)
- Prompt the user to pick an integer between 1 and 100.
- Determine if the guess matches the actual value. If it does, tell them they won the game and how many guesses they took.
- Tell the the user if they are “warmer” (if their guess is closer to the target than the previous guess), “colder” (if it is farther from the target) or “neither warmer nor colder” if their new guess is the same distance from the value as the previous guess.

- Repeat this process until the user guesses the correct number.
- Display the number of guesses it takes the user to guess the target. Feel free to write some code that comments on how well they did (i.e. "amazing!" or "better luck next time", etc.).

Key Points

- Use `if` statements to control whether a block of code is executed based on some condition
- Use `else` or `elif` statements to provide additional tests
- Python provides several ways of making and combining conditional tests
- If statements are often used inside loops
- Common combinations of if statements and loops are characterized as **Design Patterns** (or **Coding Patterns**)

This tutorial is a modified adaptation of "Python for Physicists" © Software Carpentry