Chapter 5: If Statements

BY: DR. SABEEHUDDIN HASAN

If Statements

- Programming often involves examining a set of conditions and deciding which action to take based on those conditions
- Python’s if statement allows you to examine the current state of a program and respond appropriately to that state
- A Simple Example

```
cars = ['audi', 'bmw', 'subaru', 'toyota']
for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.title())
```

If Statements

- The loop in this example first checks if the current value of car is 'bmw'
- If it is, the value is printed in uppercase
- If the value of car is anything other than 'bmw', it’s printed in title case:

```
Audi
BMW
Subaru
Toyota
```

Conditional Tests

- At the heart of every if statement is an expression that can be evaluated as True or False and is called a *conditional test*
- Python uses the values True and False to decide whether the code in an if statement should be executed
- If a conditional test evaluates to True, Python executes the code following the if statement
- If the test evaluates to False, Python ignores the code following the if statement
- ***Checking for Equality***

```
>>> car = 'bmw'
>>> car == 'bmw'
True
```

Conditional Tests

- ***Checking for Equality***
- The first line sets the value of car to 'bmw' using a single equal sign
- The next line checks whether the value of car is ‘audi' by using a double equal sign (==)
- This equality operator returns True if the values on the left and right side of the operator match, and False if they don’t match

```
>>> car = 'bmw'
>>> car == ‘audi'
False
```

Conditional Tests

- ***Ignoring Case When Checking for Equality***
  - Testing for equality is case sensitive in Python
  - Two values with different capitalization are not considered equal:

```
>>> car = 'Audi'
>>> car == 'audi'
False
```

  - If case matters, this behavior is advantageous
  - But if case doesn't matter, you can convert the variable's value to lowercase before doing the comparison

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
```

Conditional Tests

- ***Ignoring Case When Checking for Equality***
  - This test will return True no matter how the value 'Audi' is formatted because the test is now case insensitive
  - The lower() method doesn't change the value that was originally stored in car

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
>>> car
'Audi'
```

  - Websites enforce certain rules for the data that users enter in a manner similar to this
  - For example, a site might use a conditional test like this to ensure that every user has a truly unique username

Conditional Tests

- ***Checking for Inequality***
  - When you want to determine whether two values are not equal, you can use the *inequality operator* (!=)

```
requested_topping = 'mushrooms'
if requested_topping != 'anchovies':
    print("Hold the anchovies!")
```

  - This code compares the value of requested_topping to the value 'anchovies'
  - If these two values do not match, Python returns True and executes the code following the if statement

```
Hold the anchovies!
```

Conditional Tests

- ***Numerical Comparisons***
  - Testing numerical values is pretty straightforward

```
>>> age = 18
>>> age == 18
True
```

  - See the following example

```
answer = 17
if answer != 42:
        print("That is not the correct answer. Please try again!“)
That is not the correct answer. Please try again!
```

- You can include various mathematical comparisons in your conditional statements as well, such as less than **<**, less than or equal to **<=**, greater than **>**, and greater than or equal to **>=**

Conditional Tests

- ***Checking Multiple Conditions***
  - Sometimes you might need two conditions to be True to take an action
  - **Using and to Check Multiple Conditions**
    - To check whether two conditions are both *True* simultaneously, use the keyword and to combine the two conditional tests
    - If each test passes, the overall expression evaluates to *True*
    - If either test fails or if both tests fail, the expression evaluates to *False*

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 22
>>> age_0 >= 21 and age_1 >= 21
True
```

Conditional Tests

- ***Checking Multiple Conditions***
  - **Using and to Check Multiple Conditions**
    - To improve readability, you can use parentheses around the individual tests, but they are not required
    - (age_0 >= 21) and (age_1 >= 21)
  - **Using or to Check Multiple Conditions**
    - The keyword or passes when either or both of the individual tests pass
    - An or expression fails only when both individual tests fail

```
>>> age_0, age_1 = 22, 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
```

Conditional Tests

- ***Checking Whether a Value Is in a List***
  - Sometimes it’s important to check whether a list contains a certain value before taking an action

```
>>> requested_toppings = ['mushrooms', 'onions', 'pineapple']
>>> 'mushrooms' in requested_toppings
True
>>> 'pepperoni' in requested_toppings
False
```

Conditional Tests

- ***Checking Whether a Value Is Not in a List***
  - It’s important to know if a value does not appear in a list
  - You can use the keyword not in this situation

```
banned_users = ['andrew', 'carolina', 'david']
user = 'marie‘
if user not in banned_users:
        print(f"{user.title()}, you can post a response if you wish.")
```

```
Marie, you can post a response if you wish.
```

Not Operator

- The not operator in Python is a **logical operator** that negates the truth value of a condition.
- If a condition is True, not makes it False, and if it's False, not makes it True.
- It's useful when you want to reverse a condition or check if something is **not true**.
- Here are some practical examples of using the not operator:
- **Checking if a number is not positive**
  - We can use not to check if a number is **not positive**.

```
number = -5
if not number > 0:
    print("The number is not positive.")
```

  - **Output:**

```
The number is not positive.
```

Not Operator

- **Checking if a list is empty**
  - You can use not to check if a list is **empty**. An empty list evaluates to False, and not reverses it to True.

```
my_list = []
if not my_list:
    print("The list is empty.")
```

  - **Output:**

```
The list is empty.
```

- **Checking if a user is not logged in**
  - If you are creating a login system, you might want to check whether the user is **not logged in**.

```
is_logged_in = False
if not is_logged_in:
    print("User is not logged in. Please log in.")
```

  - **Output:**

```
User is not logged in. Please log in.
```

Not Operator

- **Negating a comparison**
  - You can use not to reverse the result of a comparison.

```
x = 10
if not x == 5:
    print("x is not equal to 5.")
```

  - **Output:**

```
x is not equal to 5.
```

  - One can negate other comparisons like >, >=, < and <=

Not Operator

- **Negating a Boolean flag**
  - If you have a flag that represents the state of a process, you can use not to perform an action if the flag is False.

```
task_completed = False
if not task_completed:
    print("The task is not yet completed.")
```

  - **Output:**

```
The task is not yet completed.
```

Conditional Tests

- ***Boolean Expressions***
  - A *Boolean value* is either *True* or *False*, just like the value of a conditional expression after it has been evaluated
  - Boolean values are often used to keep track of certain conditions, such as whether a game is running or whether a user can edit certain content on a website:

```
game_active = True
can_edit = False
```

  - Boolean values provide an efficient way to track the state of a program or a particular condition that is important in your program

if Statements

- ***Simple if Statements***
  - The simplest kind of if statement has one test and one action:

```
if conditional_test:
        do something
```

  - You can put any conditional test in the first line and just about any action in the indented block following the test
  - If the conditional test evaluates to True, Python executes the code following the if statement

```
age = 19
if age >= 18:
        print("You are old enough to vote!")
```

```
You are old enough to vote!
```

if Statements

- Indentation plays the same role in if statements as it did in for loops
- All indented lines after an if statement will be executed if the test passes, and the entire block of indented lines will be ignored if the test does not pass
- You can have as many lines of code as you want in the block following the if statement

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
```

```
You are old enough to vote!
Have you registered to vote yet?
```

if Statements

- ***if-else Statements***
  - Often, you'll want to take one action when a conditional test passes and a different action in all other cases
  - An *if-else* block is similar to a simple if statement, but the else statement allows you to define an action or set of actions that are executed when the conditional test fails

```
age = 17
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

```
Sorry, you are too young to vote.
Please register to vote as soon as you turn 18!
```

if Statements

- ***The if-elif-else Chain***
  - Often, you'll need to test more than two possible situations, and to evaluate these you can use Python's if-elif-else syntax
  - Python executes only one block in an if-elif-else chain
  - It runs each conditional test in order, until one passes
  - When a test passes, the code following that test is executed and Python skips the rest of the tests
  - For example, consider an amusement park that charges different rates for
  - different age groups:
    - *Admission for anyone under age 4 is free.*
    - *Admission for anyone between the ages of 4 and 18 is $25.*
    - *Admission for anyone age 18 or older is $40.*
  - For the above problem we can use *if-elif-else* chain

if Statements

- ***The if-elif-else Chain***

```
age = 12
if age < 4:
    print("Your admission cost is $0.")
elif age < 18:
    print("Your admission cost is $25.")
else:
    print("Your admission cost is $40.")
```

```
Your admission cost is $25.
```

if Statements

- *The if-elif-else Chain*

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
else:
    price = 40
print(f"Your admission cost is ${price}.")
```

if Statements

- ***Using Multiple elif Blocks***

```
age = 12
if age < 4:
        price = 0
elif age < 18:
        price = 25
elif age < 65:
        price = 40
else:
        price = 20
print(f"Your admission cost is ${price}.")
```

if Statements

- *Omitting the else Block*

```
age = 12
if age < 4:
	price = 0
elif age < 18:
	price = 25
elif age < 65:
	price = 40
elif age >= 65:
	price = 20
print(f"Your admission cost is ${price}.")
```

if Statements

- ***Testing Multiple Conditions***
- The if-elif-else chain is powerful, but it's only appropriate to use when you just need one test to pass
- As soon as Python finds one test that passes, it skips the rest of the tests
- Sometimes it's important to check all conditions of interest

```
requested_toppings = ['mushrooms', 'extra cheese']
if 'mushrooms' in requested_toppings:
        print("Adding mushrooms.")
if 'pepperoni' in requested_toppings:
        print("Adding pepperoni.")
if 'extra cheese' in requested_toppings:
        print("Adding extra cheese.")
print("\nFinished making your pizza!")
```

if Statements

- ***Testing Multiple Conditions***

```
Adding mushrooms.
Adding extra cheese.

Finished making your pizza!
```

  - This code would not work properly if we used an *if-elif-else* block, because the code would stop running after only one test passes

Using if Statements with Lists

- ***Checking for Special Items***
  - The following example displays a message whenever a topping is added to your pizza, as it’s being made

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
        print(f"Adding {requested_topping}.")
print("Finished making your pizza!")
```

```
Adding mushrooms.
Adding green peppers.
Adding extra cheese.
Finished making your pizza!
```

Using if Statements with Lists

- ***Checking for Special Items***
  - But what if the pizzeria runs out of green peppers?
  - An if statement inside the for loop can handle this situation appropriately

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
    if requested_topping == 'green peppers':
        print("Sorry, we are out of green peppers right now.")
    else:
        print(f"Adding {requested_topping}.")
print("Finished making your pizza!")
```

```
Adding mushrooms.
Sorry, we are out of green peppers right now.
Adding extra cheese.
Finished making your pizza!
```

Using if Statements with Lists

- ***Checking That a List Is Not Empty***
  - It's useful to check whether a list is empty before running a for loop

```
requested_toppings = []
if requested_toppings:
        for requested_topping in requested_toppings:
                print(f"Adding {requested_topping}.")
        print("\nFinished making your pizza!")
else:
        print("Are you sure you want a plain pizza?")
```

  - Since the list is empty so the program asks the user:

```
Are you sure you want a plain pizza?
```

Using if Statements with Lists

- *Using Multiple Lists*

```
available_toppings = ['mushrooms', 'olives', 'green peppers','pepperoni', 'pineapple', 
'extra cheese']
requested_toppings = ['mushrooms', 'french fries', 'extra cheese']
for requested_topping in requested_toppings:
    if requested_topping in available_toppings:
        print(f"Adding {requested_topping}.")
    else:
        print(f"Sorry, we don't have {requested_topping}.")
print("\nFinished making your pizza!")
```

Using if Statements with Lists

- ***Using Multiple Lists***
  - In this code, it searches for requested_toppings in the available_toppings list and prints if it is available else gives a Sorry message

```
Adding mushrooms.
Sorry, we don't have french fries.
Adding extra cheese.


Finished making your pizza!
```

Using if Statements with Lists

- **Styling Your if Statements**
  - Use a single space around comparison operators, such as ==, >=, and <=
  - For example: **if age < 4:** is better than **if age<4:**
  - Such spacing does not affect the way Python interprets your code; it just makes your code easier for you and others to read