coding temple

Python Basics

Module 2:

Lesson 2: Python Conditional Statement

Chapter 1.1:

The Magic of Conditional Statements:
The Crossroads of Python's Adventure

Welcome aspiring Pythonistas!

Have you ever stood at a crossroads, wondering which path to take? Perhaps you're on a quest, and a mysterious old signpost points in multiple directions, each leading to different outcomes. In the magical world of Python, these crossroads are nothing but conditional statements. Just as you make decisions in life based on conditions ("If it's raining, I'll stay indoors"), Python too makes decisions based on certain conditions.

coding temple

Conditional statements

'ifs', 'elses', and 'elifs'

Now, it’s time to dive deeper into this enchanted forest of 'ifs', 'elses', and 'elifs'. As you familiarize with them you realize that they are decision-making tools that allow code execution based on specific conditions.

The Mighty "If":

The First Path

Imagine standing at the entrance of a dark cave, holding a torch. If the torch is lit, you proceed; if not, you wait. In Python, this is our first path - the mighty "if". It allows you to check a condition and act upon it.

Here, if our torch is lit (torch_lit is True), we get the courage to venture into the cave.

```
torch_lit = True

if torch_lit:
        print ("Venture forth into the cave!")
```

Example 1. Simple Conditions: Clear Skies Ahead

The beauty of nature often influences our daily plans and activities. A clear sky can instantly uplift our spirits and inspire outdoor adventures. In this straightforward Python script titled "Simple Conditions: Clear Skies Ahead ☀️", we explore how a basic condition can guide our day's activity.

Here, if our torch is lit (torch_lit is True), we get the courage to venture into the cave.

```
weather = “sunny”

if weather == “sunny”:
        print (“Time for a picnic!”)
```

Example 2. Combining Conditions: Two Paths Converging

In the realm of fantasy, adventurers often find themselves at crossroads, making decisions based on the treasures they've amassed. Every coin counts, especially when it comes to acquiring magical items. In our Python script titled "Combining Conditions: Two Paths Converging 🌉", we delve into a scenario where the combined value of gold and silver coins determines the possibility of purchasing a coveted magic potion.

Here, if our torch is lit (torch_lit is True), we get the courage to venture into the cave.

```
gold_coins = 10
silver_coins = 50

if gold_coins > 5 and silver_coins > 30:
        print (“Enough to buy the magic potion!”)
```

Example 3. Not Conditions:

Beware of the Trap! 🚫

In the adventurous tales of heroism, not all foes are created equal. While some enemies can be confronted head-on, others demand caution and strategy. In our Python script titled "Not Conditions: Beware of the Trap! 🚫", we venture into a scenario where the type of enemy dictates the adventurer's approach.

```
enemy = “goblin”

if enemy != “dragon”:
        print (“Charge forward, brave adventure!”)
```

Example 4. Greater or Lesser Conditions:

The Weighing Scales of Fate ⚖️

In the intricate labyrinths of gaming, a player's health is often the thin line between victory and defeat. Every point of health can be the difference between forging ahead or facing a premature end. In our Python script titled "Greater or Lesser Conditions: The Weighing Scales of Fate ⚖️", we explore the pivotal moments where a player's health determines the next course of action.

```
player_health = 75

if player_health <= 100:
        print (“Drink a health potion for full strength!”)
```

Example 5. Checking Ranges:

The Guarded Treasure Chest

In the mystical realms of fantasy, magic stones are coveted treasures, often holding the key to unlocking hidden wonders. Collecting the right amount can grant adventurers access to ancient artifacts and secret chambers. In our Python script, we delve into a scenario where the number of magic stones in one's possession determines their eligibility to unlock the legendary "Treasure Chest".

```
magic_stones = 12

if 10 <=  magic_stones <= 20:
        print (“You've unlocked the Eleven chest!”)
```

Example 5. Combining Negative Checks

The Double Guardian 🛡️🐉

```
is_daytime = False
dragon_asleep = True

if not is_daytime and dragon_asleep:
    print("Sneak into the dragon's lair to retrieve the golden chalice!")
```

In tales of daring quests and legendary treasures, timing and circumstance often play crucial roles. Venturing into a dragon's lair is no small feat, especially when the stakes involve retrieving a priceless artifact like the golden chalice. In our Python script titled "Combining Negative Checks: The Double Guardian 🛡️🐉", we navigate the delicate balance of choosing the perfect moment to embark on this perilous mission.

is_daytime = False
dragon_asleep = True

if not is_daytime and dragon_asleep:
print (“Sneak into the dragon's lair to retrieve the golden chalice!”)

Summarizing the mighty “If”

To sum it up, the mighty "if" is like your guiding star in the vast Pythonian night sky, illuminating the path of conditions. As you traverse deeper, remember to wield its power wisely, combining its various facets to make your code shine brightly in the realm of logic.

The Elusive "Elif": The Alternate Route

Welcome, Coding templer!
Now, imagine you're walking through a mystical forest at night. As you tread carefully, you come across a crossroad with multiple signposts, each giving you a direction based on the phase of the moon.

The crucial part of the “if-elif-else” Structure

In any given `if-elif-else` structure, only one path will be chosen (we’ll talk later about else statement*) ****. Once a condition is met, Python will execute that block of code and skip the rest, even if other conditions are also true. It's like choosing a path in the forest; once you've chosen one, you can't simultaneously walk down another.

But, wait! What about, if neither the full moon nor the new moon conditions are met? you might want a default action. This is represented by the `else` statement. For now, let's set that aside and circle back to it later!

So, the entire code structure would look like this:

```
moon_phase = "full moon"

if moon_phase == "full moon":
    print("Beware of werewolves!")
elif moon_phase == "new moon":
    print("Witches' night out!")
```

The “if-elif-else” code

Step 1
Title: Set a variable called moon_phase to “full moon”.
Text: This variable represents the current phase of the moon in your story or adventure.

```
moon_phase = "full moon"
```

Step 2
Title: Start a line with an “if” statement.
Text: It's checking whether the moon_phase is equal to "full moon" using the == operator. If it is, the code block under this condition will be executed.

```
if moon_phase == "full moon":
```

Step 3
Title: Check if the variable moon_phase is “full moon”.
Text: If the moon_phase is indeed "full moon," then this line of code will be executed. It's a warning to beware of werewolves during a full moon by printing the message "Beware of werewolves!"

```
print("Beware of werewolves!")
```

The moon_phase is not "full moon"

If the moon_phase is not "full moon," we have an "elif" (short for "else if") statement to check for another condition:

```
elif moon_phase == "new moon":
```

This "elif" statement checks if the moon_phase is equal to "new moon."

If the moon_phase is "new moon," then this line of code will be executed. It's a different scenario, suggesting that it's a night when witches come out to do their magic, and it prints the message "Witches' night out!".

```
print("Witches' night out!")
```

Example 1. Potion Strength Check

In the alchemical world, the potency of a potion can determine its effectiveness, whether it's for healing, enhancing abilities, or casting spells. Crafting the perfect potion requires precision and understanding of its strength. In our Python script titled "Potion Strength Check", we evaluate the potency of a concocted brew to categorize its power.

```
potion_strength = 15

if potion_strength > 20:
    print("It's a super potent potion!")
elif potion_strength > 10:
    print("It's a moderately potent potion!")
```

How the Potion Strength Check works

Here, we're setting a variable called potion_strength to 15. This variable represents the strength or effectiveness of a magical potion you have.

```
potion_strength = 15
```

Example 2. Weather Advisory

Nature's ever-changing moods often dictate our daily plans, attire, and activities. From sunny days that beckon outdoor adventures to snowy mornings perfect for winter fun, the weather plays a pivotal role in shaping our experiences. In our Python script titled "Weather Advisory", we've crafted a handy guide that offers suggestions based on the day's forecast.

```
weather = "rainy"

if weather == "sunny":
    print("It's a bright and beautiful day!")
elif weather == "rainy":
    print("Carry an umbrella!")
elif weather == "snowy":
    print("Time to build a snowman!")
```

How the Weather Advisory works

Here, we're setting a variable called weather to "rainy." This variable represents the current weather condition in your story or adventure.

```
weather = "rainy"
```

Example 3. Magic Sword Quality

In the legends of old, the material of a sword often signifies its power, origin, and the prestige of its wielder. From gleaming golden blades that radiate royalty to ancient bronze swords that tell tales of bygone battles, each material imparts a unique aura to the weapon. In our Python script titled "Magic Sword Quality", we delve into the world of enchanted blades, determining the allure and essence of a sword based on its composition.

```
sword_material = "silver"

if sword_material == "gold":
    print("The sword shines brightly!")
elif sword_material == "silver":
    print("The sword has a mystical glimmer!")
elif sword_material == "bronze":
    print("The sword looks ancient and valuable!")
```

How the Magic Sword Quality works

Here, we're setting a variable called sword_material to "silver." This variable represents the material from which a sword is made in your adventure or story.

```
sword_material = "silver"
```

Example 4. Dungeon Level Access

In the vast realms of fantasy gaming, a player's level is a testament to their skills, experience, and the challenges they've overcome. As players progress, they earn the right to venture into more perilous and rewarding dungeons, each tailored to match their prowess. In our Python script titled "Dungeon Level Access", we navigate the gates of these mysterious labyrinths, determining which dungeon a player is qualified to enter based on their level.

```
player_level = 12

if player_level < 5:
    print("Access the beginner dungeon!")
elif 5 <= player_level < 10:
    print("Enter the intermediate dungeon!")
elif player_level >= 10:
    print("Challenge the advanced dungeon!")
```

How the Dungeon Level Access works

Here, we're setting a variable called **player_level** to 12. This variable represents the level or experience of the player in your game or adventure.

```
player_level = 12
```

Example 5. Dragon's Lair Alert System

In the world of high fantasy, dragon lairs are places of wonder and danger, often hiding immense treasures guarded by these majestic creatures. Venturing into such a lair requires both bravery and wisdom, as the presence of a dragon or the allure of hidden treasure can change the course of an adventurer's journey. In our Python script titled "Dragon's Lair Alert System", we've designed a mechanism to gauge the risks and rewards of entering a dragon's domain.

```
is_dragon_present = True
has_treasure = False

if is_dragon_present and not has_treasure:
    print("Enter with caution! Dragon ahead, but no treasure in sight.")
elif not is_dragon_present and has_treasure:
    print("No dragon around! Quick, grab the treasure.")
elif is_dragon_present and has_treasure:
    print("A mighty dragon guards the treasure! Tread carefully.")
else:
    print("Empty lair. Safe to explore, but no treasures here.")
```

How the Dragon's Lair Alert System works

Here, we have two variables:

- is_dragon_present is set to True, indicating that there is a dragon in the area.
- has_treasure is set to `False, suggesting that there is no treasure nearby.

These variables help describe the situation in your adventure.

```
is_dragon_present = True
has_treasure = False
```

Example 6. Potion Mixing Consequences

Alchemy is an art of precision and experimentation, where the combination of ingredients can lead to wondrous or sometimes unintended results. In the mystical world of potion-making, each ingredient has its unique properties, and mixing them can either enhance their effects or lead to volatile reactions. In our Python script titled "Potion Mixing Consequences", we delve into the outcomes of combining the red and blue potions.

```
red_potion = True
blue_potion = False

if red_potion and not blue_potion:
    print("You get a potion of strength!")
elif not red_potion and blue_potion:
    print("You get a potion of speed!")
elif red_potion and blue_potion:
    print("Oops! Mixing red and blue makes it explode!")
else:
    print("No potion was mixed.")
```

How the Potion Mixing Consequences works

Here, we have two variables:

- red_potion is set to True, indicating that you have a red potion.
- blue_potion is set to False, suggesting that you don't have a blue potion.

These variables represent the types of potions you have in your adventure.

```
red_potion = True
blue_potion = False
```

The Trusty "Else": When All Else Fails

Welcome back, fellow Coding Templer!

Every seasoned adventurer knows the importance of preparation. As you journey through unknown terrains, you'll often come across multiple paths, each with its own set of challenges and decisions. However, there will be times when none of the paths seem right, or the conditions to proceed are not met. In such situations, you need a reliable backup plan, a safety net that ensures you're never truly lost. This is where the trusty "else" comes into play.

What you need to know about the “else” statement

In the world of Python, the else statement acts as this backup plan. After checking various conditions using if and elif, if none of them are satisfied, the else block provides a default action to be taken. It's like the compass in an adventurer's toolkit, always pointing you in a direction when all other paths are uncharted or inaccessible.

When no path is accessible

Let’s consider this scenario: You're in a dense forest, and you have a map that shows two paths. One path is accessible only when it's daytime, and the other path is accessible only when it's raining. In Python, this can be represented as:

```
if is_daytime:
    print (“Take the sunny path through the meadow!”)
elif is_raining:
    print (“Take the rainy path through the marsh!”)
```

But what if it's neither daytime nor raining? What if you're in the forest at dusk and the sky is clear? This is where our trusty "else" comes into play:

```
else:
    print (“Neither path is suitable right now. Use your compass and head back home!”)
```

The else statement ensures that there's always a course of action, a default response when all other conditions fail. It's the adventurer's assurance that no matter how uncertain or challenging the situation, there's always a way out, a direction to follow.

The fundamental rules of the “else” statement

As our intrepid adventurer navigates the intricate pathways of the forest, it's essential to remember the fundamental rules of the journey. In the realm of Python, the else statement, while a reliable fallback, cannot stand alone. It always requires a preceding if or elif to set the conditions. Think of it this way: before resorting to your compass, you first check the map (if) and then perhaps the weather (elif). Only when these primary guides are inconclusive do you rely on your compass (else).

In summary, the entire decision-making structure would look like this:

```
is_daytime = False
is_raining = False
if is_daytime:
    print (“Take the sunny path through the meadow!”)
elif is_raining:
    print (“Take the rainy path through the marsh!”)
else:
    print (“Neither path is suitable right now. Use your\
compass and head back home!”)
```

But what if it's neither daytime nor raining? What if you're in the forest at dusk and the sky is clear? This is where our trusty "else" comes into play:

```
else:
    print (“Neither path is suitable right now. Use your \
compass and head back home!”)
```

The else statement ensures that there's always a course of action, a default response when all other conditions fail. It's the adventurer's assurance that no matter how uncertain or challenging the situation, there's always a way out, a direction to follow.

Example 1. Weather Wardrobe Guide

As you know, dressing appropriately for the weather can make all the difference in our day. No matter if it’s a sunny afternoon or a chilly rainy morning, our wardrobe choices can ensure we remain comfortable and prepared. In this Python script, we've created a handy Weather Wardrobe.

```
is_raining = True
is_cold = False

if is_raining and is_raining:
   print (“Wear a waterproof jacket and scarf!”)
elif is_raining:
   print (“Don't forget your umbrella!”)
else:
   print (“Looks like a clear day, dress as you wish!”)
```

How the Weather Wardrobe Guide works

Here, we have two variables represent the weather conditions in your scenario:

- is_raining is set to True, indicating that it's currently raining.
- is_cold is set to False, suggesting that it's not cold.

```
is_raining = True
is_cold = False
```

Example 2. Monthly Savings Calculator

Managing finances and tracking savings is a crucial aspect of achieving our financial goals. Whether you're saving up for a special purchase, an emergency fund, or just looking to have a better grasp on your monthly finances, understanding your savings is key. In this Python script, we introduce a simple Monthly Savings Calculator.

```
income = 5000 # Monthly income
expenses = 4500 # Monthly income

savings = income - expenses

if savings > 1000:
   print ("Great job! You saved a lot this month.")
elif savings <= 0:
   print ("Looks like you've spent all or more than you \
earned! ")
else:
   print ("Every little bit counts! Keep saving.")
```

How the Monthly Savings Calculator works

Here, we have two variables which help you track your financial situation for the month:

- income is set to 5000, representing your monthly income.
- expenses is set to 4500, representing your monthly expenses.

```
savings = income - expenses
```

This line calculates your savings for the month by subtracting your expenses from your income. In this case, you have:

- income of 5000
- expenses of 4500

So, your savings will be 5000 - 4500, which equals 500.

Example 3. Museum Entry Discounts

Museums are wonderful places of learning and exploration, and many offer special discounts to encourage a diverse range of visitors. Whether you're a student immersed in studies or a senior enjoying the golden years, there might be a discount waiting for you! In this Python script, we've designed a Museum Entry Discounts system.

```
is_student = True
is_senior = False

if is_student:
   print ("You get a 50% student discount!")
elif is_senior:
   print ("Seniors enjoy a 40% discount! ")
else:
   print ("Regular entry fee applies.")
```

How the Museum Entry Discounts works

Here, we have two variables that represent your status as a student and whether you are a senior citizen.

```
is_student = True
is_senior = False
```

- is_student is set to True, indicating that you are a student.
- Is_senior is set to False, suggesting that you are not a senior citizen.

This code then checks these variables to determine the discount you might be eligible for.

Example 4. Weekend Activity Planner

Weekends are the perfect time to relax, recharge, and indulge in activities we love. But sometimes, the weather and our budget can influence our plans. Whether it's a sunny day beckoning outdoor adventures or a tight budget prompting creative indoor activities, making the most of the weekend is all about smart planning. In this Python script, we've crafted a Weekend Activity Planner.

```
is_sunny = True
have_money = False

if is_sunny and not have_money:
    print ("Great day for a walk in the park!")
else:
    print ("Maybe consider indoor activities or saving for a \
sunny day outing.")
```

How the Weekend Activity Planner works

Here, we have two variables that represent the weather condition and your financial situation.

```
is_sunny = True
have_money = False
```

- is_sunny is set to True, indicating that the weather is sunny.
- have_money is set to False, suggesting that you don't have any money with you at the moment.

Example 5. Game Character Interaction

The immersive world of video games makes us interact with various characters that can lead us to unexpected adventures, challenges, or even friendships. Each character you encounter might have a unique role, from offering quests to simply being a part of the game's vibrant environment. In this Python script, we delve into a Game Character Interaction system.

```
is_friendly = False
has_quest = True

if not is_friendly:
        print ("Be cautious! This character might not be helpful.")
elif has_quest:
        print ("This character has a quest for you!")
else:
        print ("Just a regular villager passing by.")
```

How the Game Character Interaction works

In this code, we have two variables that represent the character's disposition and whether they have a quest to offer.

- is_friendly is set to False, indicating that the character you're encountering might not be friendly.
- has_quest is set to True, suggesting that the character has a quest for you.

```
is_friendly = False
has_quest = True
```

Example 6. Drive-Thru Order Suggestion

Fast food drive-thrus provide a variety of choices to suit different tastes and preferences. Whether you're interested in vegetarian options, looking for something spicy, or simply exploring the menu, there's always a tasty choice available. The Python script we've created here is a Drive-Thru Order Suggestion system, designed to help guide you to a delicious selection based on your preferences.

```
wants_veggie = True
likes_spice = False

if wants_veggie and likes_spice:
    print ("How about a spicy veggie wrap?")
elif wants_veggie:
    print ("Try our classic veggie burger!")
else:
    print ("check out our grill menu!")
```

How the Game Drive-Thru Order Suggestion works

Here, we have two variables that represent the customer's food preferences.

- wants_veggie is set to True, suggesting that the person you're serving prefers a vegetarian option.
- likes_spice is set to False, indicating that the person does not like spicy food.

```
wants_veggie = True
likes_spice = False
```

Conclusion

In this Chapter 1, you have tackled:

- The use of the conditional statement "**if**" as a first path.
- The use of the conditional statement "**elif**" as an alternative path
- The use of the conditional statement "**else**" when all else fails.

Practice with your own exercises and apply all the concepts developed in this chapter.

Decisions at the Crossroad

Task 1: Code Correction
You are provided with a Python script that uses conditional statements to tell if a number is positive, negative, or zero, but it has some errors. Identify and fix them.

Buggy Code

```
number = input("Enter a number: ")

if number > 0:
    print("The number is positive.")
elif number = 0:
    print("The number is zero.")
else number < 0:
    print("The number is negative.")
```

The Story Brancher

Task 1: Code Correction

You are provided with a Python script that uses "`if-else`" structures for a story branching mechanism. There are some errors in the code. Identify and correct them.

Buggy Code

```
choice = input("Do you choose the path to the left or right? ")

if choice = "left":
    print("You find a treasure chest!")
elif choice = "right":
    print("You encounter a dragon!")
else:
    print("Invalid choice!")
```

The Greatest Showdown

Objective:
Harness the power of conditional statements to compare and determine values.
Tasks:
Task 1: Identify the Greatest
Write a Python program that prompts the user to enter three numbers. The program should then identify and print out the largest number among the three.
Task 2: Identify the Smallest
Extend your program from Task 1 to also determine the smallest number among the three and print it out.
Task 3: Equality Check
Enhance your program to handle cases where two or all of the numbers are equal. The program should display appropriate messages like "Two numbers are equal and the largest" or "All numbers are equal".

Leap Year Explorer

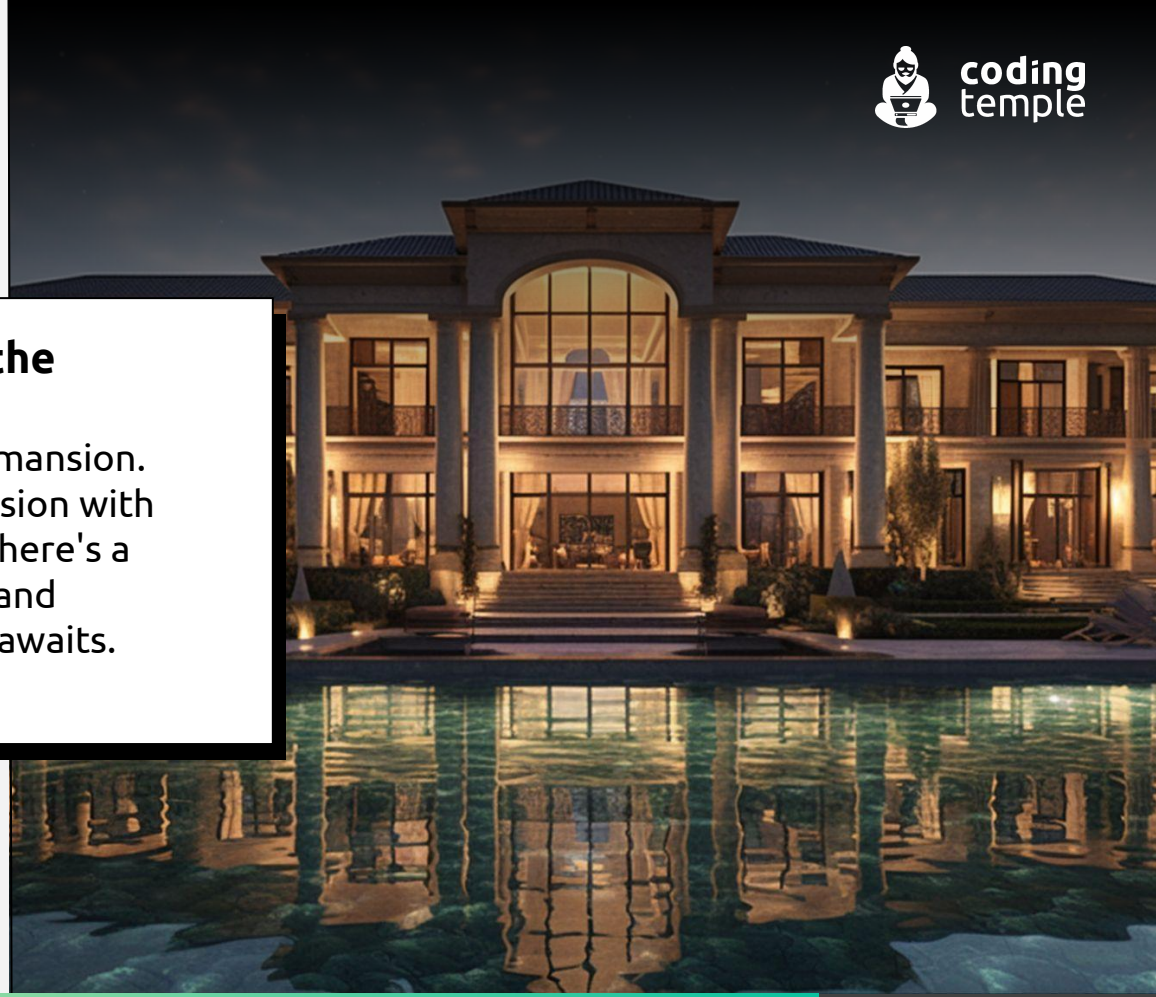
Objective:
Dive deep into the intricacies of the calendar by exploring the concept of leap years.
Tasks:
Task 1: Leap Year Checker
Write a Python program that prompts the user to input a year. The program should determine if the given year is a leap year or not and then display an appropriate message.
Task 2: Century Verification
Add functionality to your program from Task 1 to inform the user if the entered year is a century year (e.g., 1900, 2000) regardless of whether it's a leap year or not.
Task 3: Time Traveler
Enhance your program to indicate if the provided year is in the future, past, or is the current year, compared to the system's current year. You might find Python's datetime module helpful for this task.

- These assignments prompt students to construct code from scratch while applying their knowledge of conditional statements. Each task within the assignments builds upon the previous, guiding students to progressively enhance their solutions.
- The errors in the "Buggy Code" sections are intentional to challenge students to identify and correct them. The "Corrected Code" sections show how the code should be properly written.

Nested If: The Mansion with Multiple Rooms

In this chapter, we'll explore the realm of decisions.

Now, imagine walking into a grand mansion. It's not just any mansion - it's a mansion with rooms inside rooms. As you enter, there's a decision to be made at every door, and within that room, another decision awaits. This is the essence of a Nested If.



Explaining this with an example

Take the weather, for instance—it's full of surprises. A day that seems mildly cool can swiftly transform into a chilly one, catching us off guard. To tackle this, we examine the current temperature. Initially, we offer broad advice based on this assessment. However, if the conditions turn colder than expected, we go a step further, giving more detailed guidance. Now, let's explore the code together and witness how it assists users in managing those unpredictable weather days, ensuring they stay comfortably warm.

```
temperature = 20

if temperature < 25:
    print("It's a bit chilly!")
if temperature < 15:
    print("Actually, you might need a coat.")
```

How the Nested If works

Here, we have a variable temperature set to 20, representing the current temperature in degrees Celsius.

```
temperature = 20
```

This line starts an "if" statement. It's checking if the temperature is less than 25 degrees Celsius.

```
if temperature < 25:
```

If the temperature is indeed less than 25 degrees Celsius, this line of code will be executed. It informs you that it's a bit chilly, and it prints the message "It's a bit chilly!"

```
print("It's a bit chilly!")
```

Nevertheless, the code doesn't end here. It continues to evaluate the temperature within the "if" statement which is checking if the temperature is less than 15 degrees Celsius.

```
if temperature < 15:
```

If it’s (which means it's even colder), this line of code will also be executed giving you an idea that you might need a coat due to the colder temperature, and it prints the message "Actually, you might need a coat."

```
print("Actually, you might need a coat.")
```

Example 1. Checking Age Restrictions

Age limitations are widespread in different situations, encompassing activities like driving, voting, and the consumption of specific goods. Confirming that individuals satisfy the stipulated age requirements is vital for both safety and legal compliance. Within this Python script, we've devised a tool to check whether someone meets the age criteria for driving, voting, or consuming alcoholic beverages, particularly in locations where the legal drinking age is set at 21.

```
age = 17
if age >= 18:
        if age >= 21:
                print("You can drive, vote, and also drink!")
        else:
                print("You can drive and vote!")

else:
        print("You're too young to drive or vote.")
```

Example 2. Weekend Activities Planner

The weekends offer an opportunity for relaxation, enjoyment, and leisure. Depending on the time of day, our activity preferences may vary. This Python script assists in directing our weekend decisions by considering the current day and time, ensuring we optimize our leisure time.

```
day = "Saturday"
time = "Morning"

if day == “Saturday”:
   if time == “Morning”:
      print("Time for cartoons!”)
   elif time == “Evening”:
      print(”Maybe a movie night?")
   else:
      print("Relax and enjoy your day!")
```

How the Weekend Activities Planner works

A weekend activity planner in Python typically works by evaluating the current day and time and suggesting activities based on predefined criteria. Here, we have two variables:

- day represents the day of the week.
- time indicates the time of day.

Example 3. The Library: Book Genre and Author 📚

Libraries hold a wealth of stories, information, and exciting journeys. The type of book and its writer can whisk readers away to various realms, eras, and encounters. This Python script assists in managing readers' expectations by analyzing their selected book genre and author.

```
genre = "Fantasy"
author = "3.K. Rowling”
        if genre == "Fantasy"
                if author == "3.K. Rowling”
                print(“Ah, an adventure at Hogwarts!")
        elif author == “R.R. Tolkien”
                print("Time to visit Middle-Earth!")
        else:
        print(“Fantasy is a journey to another world!”)
```

How the Library: Book Genre and Author 📚 works

To create a Python script that works as a library to manage readers' expectations based on book genre and author, you can structure the program with functions and use data structures like dictionaries.
Here, we have two variables:

- genre represents the genre of the book.
- author indicates the author of the book.

Example 4. The Kitchen: Fruit and Ripeness 🍎

The kitchen is like a magical place for cooking, and picking the right fruit at the perfect ripeness is crucial for making tasty dishes. This Python script helps figure out if a fruit is ready to use by considering its type and how ripe it is.

```
fruit = "Apple"
is_ripe = True
has_spots = False
if fruit == "Apple":
    if is_ripe and not has_spots:
        print("Perfect for a juicy bite!")
    elif not is_ripe and not has_spots:
        print("Let it ripen a bit more.")
    else:
        print("Might be best for apple pie!")
elif fruit == "Banana":
    if is_ripe and not has_spots:
        print("Ready to eat!")
    elif not is_ripe:
        print("Still a bit green.")
    else:
        print("Perfect for banana bread!")
else:
    print("Not sure about this fruit's ripeness.")
```

How the Kitchen: Fruit and Ripeness 🍎 works

Creating a Python script to assess the state of a fruit based on its type and ripeness involves defining conditions and logic to guide users.
Here, we have three variables which will be used to guide the culinary decision.

- fruit represents the type of fruit.
- Is_ripe is a boolean indicating if the fruit is ripe.
- Has_spots is a boolean indicating if the fruit has spots, which might suggest over-ripeness or damage.

Example 5. The Cinema Room: Movie and Genre 🎥

The cinema room, a magical space where narratives unfold on the expansive screen. The selection of a movie and its genre has the power to shape the ambiance of the night. Whether you crave an exhilarating action-packed scene or a touching romantic tale, this Python script assists in navigating the cinematic journey according to the chosen movie and its genre.

```
movie = "Inception"
release_year = 2010  # Using an integer to represent the year
duration_minutes = 148  # Using an integer for the movie's duration

if movie == "Inception":
    if 2000 <= release_year <= 2020 and duration_minutes > 120:
        print("A modern classic with a runtime over 2 hours!")
    elif release_year < 2000:
        print("A gem from the past!")
    else:
        print("A recent masterpiece!")
elif movie == "Titanic":
    if release_year == 1997 and duration_minutes > 180:
        print("A romantic epic that spans over 3 hours!")
    else:
        print("A timeless love story!")
else:
    print("Enjoy the movie!")
```

How the Cinema Room: Movie and Genre 🎥 works

The Cinema Room: Movie and Genre 🎥 operates by providing a tailored cinematic experience based on your preferences. Just like a skilled conductor orchestrating a symphony, this system harmonizes the elements of film and genre to set the perfect mood for your movie night.

Here, we have three variables:

- movie represents the title of the movie.
- release_year is an integer indicating the year the movie was released.
- duration_minutes is an integer representing the movie's duration in minutes.

Shorthand If: The Quick Texting Lingo

In this chapter, we'll explore shortcuts.

In today's fast-paced society, we embrace efficiency. Why bother typing "Be right back" when "BRB" does the job? Python understands this and provides the If shorthand, allowing for swift decision-making on the fly. It called a ternary operation

Explaining the Shorthand If's code

```
x, y = 5, 10
print ("x is greater") if x > y else print("y is greater")
```

Here, we have two variables, x and y, which are assigned the values 5 and 10, respectively.

```
x, y = 5, 10
```

The latest part of the code, uses a conditional expression (also known as a ternary operator) to determine which message to print based on a condition.

```
print ("x is greater") if x > y else print("y is greater")
```

How the ternary operator works

Here we have two conditions being evaluated x > y, which checks if the value of x is greater than the value of y. In this case, 5 is not greater than 10, so this condition evaluates to False.
The syntax checks if x is greater than y.
If the condition is True, it executes the expression print ("x is greater").
If the condition is False, it executes the expression after else, which is print("y is greater").
In the given example, since 5 is not greater than 10, the condition is False, and the code prints "y is greater."

Example 1. Quick Weather Check

In the Python script, we've created a brief guide for making activity decisions based on the weather, reflecting the common practice of tailoring plans to weather conditions—choosing beach activities for sunny days and indoor pursuits for gloomy ones.

```
weather = "sunny"
activity = "beach" if weather == "sunny" else "indoor games"
print(activity)  # Outputs: beach
```

The ternary expression for Quick Weather Check

Here, we have the variable weather which is set to “sunny”, indicating the current weather condition and the scenario for your day.

```
activity = "beach" if weather == "sunny" else "indoor games"
```

This line uses the shorthand if statement. It's checking a single condition:
weather is "sunny", meaning the day is bright and sunny.
The shorthand if is asking, "Is the weather sunny?" If this condition is true (it's a sunny day), then the activity is set to "beach". Otherwise, the activity is set to "indoor games".

Example 2. Quick Beverage Choice

Now, let's delve into a detailed explanation for the "Quick Beverage Choice" example, incorporating numbers, logical operators, and the shorthand if:
The beverages we choose are often shaped by the time of day. For instance, we might opt for a caffeinated drink in the morning for an energy boost, while in the evening, a soothing tea might be more fitting. This Python script serves as a brief guide, providing recommendations on selecting beverages based on the current time of day and energy levels.

```
hour_of_day = 7  # Using an integer to represent the hour in 24-hour format
energy_level = 3  # Using an integer on a scale of 1 to 5, where 5 is very energetic

beverage = "coffee" if (6 <= hour_of_day < 12) and energy_level < 4 else "tea"
print(beverage)  # Outputs: coffee
```

The ternary expression for Quick Weather Check

A Quick Beverage Choice ☕ in Python typically works by representing the current time and how you feel.
Here, we have two variables:

- hour_of_day is set to 7, indicating it's 7 AM.
- energy_level is set to 3, suggesting a moderate level of energy.

```
beverage = "coffee" if (6 <= hour_of_day < 12) and energy_level < 4 else "tea"
```

On this occasion, the shorthand if statement. It's checking two conditions:

- The time is between 6 AM and 12 PM (morning hours).
- Your energy level is below 4.

The shorthand if is asking, "Is it morning and is your energy level moderately low?" If both conditions are true, then the beverage is set to "coffee". Otherwise, the beverage is set to "tea".

Example 3. Quick Workout Routine

The Python script serves as a personalized fitness guide, helping you select an appropriate workout by considering your energy level and available time. Reflecting the impact of fluctuating energy levels and busy schedules on workout preferences, the script offers tailored recommendations, providing a quick and efficient way to decide on a fitness routine that suits both your current energy level and time constraints.

```
energy_level = 4.5  # Using a float on a scale of 1.0 to 5.0, where 5.0 is very energeti
c
time_available = 30.5  # Using a float to represent minutes available for workout
short_on_time = time_available < 45.0

workout = "intense cardio" if energy_level > 4.0 and not short_on_time else "light yoga"
print(workout)  # Outputs: light yoga
```

How the “Shortland If” works with the Quick Workout Routine

A Quick Workout Routine 🏋️ in Python works by representing how you feel and how much time you can dedicate to a workout. Here, we have two variables:
energy_level is set to 4.5, indicating a high level of energy.
time_available is set to 30.5, suggesting you have 30.5 minutes available.

```
short_on_time = time_available < 45.0
```

This line evaluates whether you're short on time for a workout. If you have less than 45 minutes, it sets short_on_time to True.

The “Shortland if” statement for a Quick Workout Routine

On this occasion, the shorthand if statement. It's checking two conditions:

- Your energy level is above 4.0.
- You're not short on time

```
workout = "intense cardio" if energy_level > 4.0 and not short_on_time else "light yoga"
```

The shorthand if is asking, "Do you have a high energy level and enough time?" If both conditions are true, then the workout is set to "intense cardio". Otherwise, the workout is set to "light yoga".
In the event the energy level is high, due to the constraint on time, it suggests light yoga and prints the message "light yoga".

```
print(workout)
```

Example 4. Quick Meal Choice

Choosing what to eat can be tricky, especially considering both the time of day and how hungry you are. It's like figuring out if you want a light snack in the middle of the day or a more filling dinner in the evening. To make this decision easier, we created a quick guide using a Python script. This guide considers the current time and how hungry you are to suggest a meal that fits the moment. It's like having a helpful assistant to make sure you pick a meal that suits both your schedule and your appetite.

```python
current_hour = 15  # Using an integer to represent the hour in 24-hour format
hunger_level = 7  # Using an integer on a scale of 1 to 10, where 10 is extremely hungry

meal = "snack" if current_hour < 17 and hunger_level < 5 else "full meal"
print(meal)  # Outputs: full meal
```

The ternary expression for the Quick Meal Choice

On this occasion, the variables current_hour and hunger_level are representing the current time and how hungry you are:
current_hour is set to 15, indicating it's 3 PM.
hunger_level is set to 7, suggesting a high level of hunger.

```
current_hour = 15
hunger_level = 7
```

In this type of code, the shorthand if statement. It's checking two conditions:

- The current hour is before 5 PM.
- Your hunger level is below 5.

The shorthand if is asking, "Is it before dinner time and are you not very hungry?" If both conditions are true, then the meal is set to "snack". Otherwise, the meal is set to "full meal" as in the example it's 3 PM.

```
print(meal)
```

Example 5. Quick Vacation Spot

Deciding where to go for a vacation can be hard, especially when you want to find a place you like and that fits your budget. It's a bit like choosing what game to play or what movie to watch. In this Python script, we made a helpful guide that considers two things: whether you love the beach and how much money you have to spend. It's like having a friendly advisor to quickly tell you a great vacation spot that matches both what you love and what you can afford.

```
loves_beach = True
budget = 1500  # Initial budget in dollars
high_budget = budget >= 2000

destination = "beach resort" if loves_beach and not high_budget else "luxury mountain re
sort"
budget -= 500 if destination == "beach resort" else 1000  # Compound assignment

print(destination)  # Outputs: beach resort
print("Remaining budget:", budget)  # Outputs: Remaining budget: 1000
```

How the ternary expression with the Quick Vacation Spot

A Quick Workout Routine 🏋️ in Python works by representing your vacation preferences and budget.
Here, we have two variables:

- loves_beach is set to True, indicating a preference for the beach.
- budget is set to 1500, representing the available budget in dollars.

```
high_budget = budget >= 2000
```

This line evaluates whether you're short on time for a workout. If your budget is 2000 dollars or more, high_budget is set to True.

The ternary expression for a Quick Vacation Spo

On this occasion, the shorthand if statement. It's checking two conditions:

- You love the beach.
- You don't have a high budget.

```
destination = "beach resort" if loves_beach and not high_budget else "luxury mountain re
sort"
```

The shorthand if is asking, "Do you love the beach and have a moderate budget?" If both conditions are true, then the destination is set to "beach resort". Otherwise, the destination is set to "luxury mountain resort".
Depending on the chosen destination, it deducts the cost of the vacation from your budget. If the destination is a beach resort, 500 dollars are deducted. Otherwise, 1000 dollars are deducted for the luxury mountain resort.

```
budget -= 500 if destination == "beach resort" else 1000
```

In the event the beach is preferred and the budget is moderate, it suggests a beach resort and prints the message "beach resort". The remaining budget after the deduction is also printed.

```
print(destination)
print("Remaining budget:", budget)
```

Example 6. Quick Study Method

Deciding how to study is like choosing the right game to play. If you're getting ready for a test or just revisiting a topic, this Python script acts like a game guide, suggesting the best way to study based on how hard the topic is and how much time you have. It's like having a friendly advisor for your study sessions, making it easier to pick the most effective method for your learning needs.

```
topic_difficulty = "hard"
available_hours = 3.5  # Using a float to represent hours available for study
understanding_level = 6  # Using an int on a scale of 1 to 10, where 10 is full understa
nding

study_method = "deep dive" if topic_difficulty == "hard" and available_hours > 2.5 else
"quick review"
bonus_hours = 1.5 if understanding_level < 5 else 0.5  # Compound assignment
available_hours += bonus_hours  # Add bonus hours to available hours

print(study_method)  # Outputs: deep dive
print("Total study hours:", available_hours)  # Outputs: Total study hours: 4.0
```

How the ternary expression works with the Quick Study Method

The ternary expression in Python is a concise way to write conditional statements and could be used to quickly determine the recommended study method based on certain conditions.

```
topic_difficulty = "hard"
available_hours = 3.5
understanding_level = 6
```

Here, we have three variables which represent the topic's difficulty, your available study time, and your current understanding.

- topic_difficulty is set to "hard", indicating the topic is challenging.
- available_hours is set to 3.5, representing the hours you can dedicate to studying.
- understanding_level is set to 6, suggesting a moderate understanding of the topic.

The ternary expression for the Quick Study Method

In this case, the shorthand if statement is checking two conditions:

- The topic difficulty is hard.
- You have more than 2.5 hours available.

```
destination = "beach resort" if loves_beach and not high_budget else "luxury mountain re
sort"
```

The shorthand if is asking, "Is the topic hard and do you have sufficient time?" If both conditions are true, then the study method is set to "deep dive". Otherwise, the study method is set to "quick review".

```
study_method = "deep dive" if topic_difficulty == "hard" and available_hours > 2.5 else
"quick review"
```

If your understanding level is below 5, you get an additional 1.5 hours. Otherwise, you get an extra 0.5 hours.

```
available_hours += bonus_hours
```

Finally, it's also possible to use a compound assignment which adds the bonus hours to your available study hours.

```
print(study_method)
print("Total study hours:", available_hours)
```

Conclusion

- This is the end of the chapter through which:
- We explored how the shorthand `if` can be effectively used in various real-world scenarios, from deciding on vacation spots to choosing study methods.
- We combined the shorthand `if` with different data types like strings, integers, and floats, showcasing its versatility.
- We integrated the shorthand `if` with logical operators and compound assignments, demonstrating its compatibility with other Python constructs.

Chapter 2.3:

The Pass Statement: The Poised Actor Awaiting Direction 🎭

In this chapter, we'll get to know about the pass statement.
Picture this: An actor on stage, spotlight on them, but they don’t have a line just yet. They wait in position, making their presence known without action. In Python, the pass statement is this actor. It's a placeholder, a statement that does...well, nothing!

Why use the pass statement?

If you;ve established the framework of your code but haven't finalized the details yet, you might hesitate to leave it entirely empty as Python doesn't appreciate that. Instead, you can employ the pass statement to maintain the code's functionality temporarily while you come back later to fill in the actual content.

Example 1. The Unfinished Function

In the expansive realm of programming, we often plan the framework of our code, preparing for future development. The Python pass statement serves as a placeholder, enabling us to sketch out functions, loops, or classes without providing immediate specifics. It's comparable to a bookmark, indicating a position in the code that we plan to revisit later.

```
def to_be_defined():
    pass

to_be_defined()  # No output and no error
```

Example 2. The Silent Exception Handler 🤫

In the realm of programming, errors are inevitable. They're like unexpected plot twists in a gripping novel. However, not all errors need to halt our story.
Sometimes, we anticipate them and choose to handle them gracefully, allowing our script to continue its narrative. The pass statement in Python can be a silent guardian in such scenarios, ensuring that certain exceptions don't disrupt the flow of our tale.

Example 3. The Placeholder Loop

In the vast world of programming, loops can be compared to repetitive scenes in a play, unfolding multiple times with subtle differences. There are instances during scripting when we recognize the necessity of a loop, but we're not prepared to specify its actions. We may be awaiting inspiration or additional data. During these moments, the pass statement acts as a quiet stand-in, guaranteeing the continuation of the program without any disruptions.

```
for i in range(5):
    pass

print("Loop completed!")
```

Example 4. The Yet-To-Be-Defined Class 🏛️

In object-oriented programming, classes serve as blueprints for creating objects, outlining their structure, properties, and behaviors. However, during the design phase, we may conceptualize the class structure without specifying all the details, especially for methods. The pass statement acts as a placeholder in such cases, enabling us to establish the class framework while deferring the definition of method actions until a later stage.

```python
class FutureImplementation:
    def method_one(self):
        pass

    def method_two(self):
        pass

obj = FutureImplementation()
obj.method_one()  # No output and no error
```

Example 5. The Conditional Placeholder 🤷

In the complex choreography of programming, conditionals are crucial points where decisions shape the narrative flow. During scriptwriting, there are instances when we know the general direction at a programming crossroads but lack specifics for the journey. In these moments, the pass statement serves as a quiet placeholder, marking an unexplored path yet to be defined.

```python
value = 10

if value > 5:
    pass  # Will implement this later
else:
    print("Value is not greater than 5.")
```

Example 6. The Review Rating: Amazon Products and Stars 🌟

The landscape of shopping has been transformed by online platforms, fundamentally altering the way we acquire goods. In this paradigm, the significance of reviews and ratings cannot be overstated in influencing our choices as consumers. Platforms like Amazon utilize star ratings as a succinct indicator of a product's quality and the satisfaction of its customers. This Python script has been developed to assist users in comprehending the implications of star ratings on Amazon products, facilitating well-informed decisions during the purchasing process.

```
product = "Kindle Paperwhite"
average_rating = 4.7  # Using a float to represent the average star rating
number_of_reviews = 25000  # Using an integer for the number of reviews

if product == "Kindle Paperwhite":
    if average_rating >= 4.5 and number_of_reviews > 10000:
        print("A highly recommended product with numerous positive reviews!")
    elif average_rating >= 4.0 and number_of_reviews > 5000:
        print("A well-received product with a good number of reviews.")
    elif average_rating < 4.0:
        print("The product has mixed reviews. Consider reading detailed reviews before p
urchasing.")
    else:
        print("The product has a high rating but lacks a significant number of review
s.")

elif product == "Amazon Echo":
    if average_rating >= 4.5 and number_of_reviews > 15000:
        print("A top-rated smart speaker with a vast number of positive reviews!")
    elif average_rating >= 4.0 and number_of_reviews > 7000:
        print("A popular smart device with many satisfied customers.")
    elif average_rating < 4.0:
        print("The smart speaker has varied reviews. It might be worth checking out newe
r models.")
    else:
        print("The Echo has a commendable rating, but more reviews would provide better
clarity.")

elif product == "Amazon Fire Stick":
    if average_rating >= 4.5 and number_of_reviews > 20000:
        print("A must-have for streaming enthusiasts with overwhelming positive feedbac
k!")
    elif average_rating >= 4.0 and number_of_reviews > 8000:
        print("A favorite among many for streaming, with a good number of reviews.")
    elif average_rating < 4.0:
        print("The Fire Stick has mixed feedback. Consider other streaming options or ne
wer models.")
    else:
        print("The Fire Stick seems promising, but more reviews could offer a clearer pi
cture.")

else:
    print("Please check the product's detailed reviews and ratings on Amazon.")
```

How the Review Rating: Amazon Products and Stars 🌟 works

The Review Rating: Amazon Products and Stars 🌟 functions as a virtual guide, much like an insightful literary critic. By allowing users to input their preferences, the Python script decodes the significance of star ratings on Amazon products. It provides a swift overview of a product's quality and customer satisfaction, akin to a critic interpreting a novel's rating. This digital guide empowers users to make well-informed purchasing decisions in the expansive online marketplace, guiding them through virtual aisles to select products that meet their expectations. Here, we have three variables:

- product represents the name of the product on Amazon.
- average_rating is a float indicating the average star rating of the product.
- number_of_reviews is an integer representing the total number of reviews the product has received.

Conclusion

Nested if statements in Python offer a robust tool for decision-making by allowing the creation of complex structures that evaluate multiple conditions hierarchically. This approach provides refined control over program flow, demonstrated through exercises like weather-based wardrobe choices and online shopping decisions based on product ratings. These real-world scenarios showcase the versatility and practicality of nested if statements, demonstrating how they handle multiple layers of information, improving code organization, readability, and efficiency. In summary, mastering nested if statement is fundamental in programming, enhancing the logic and functionality of applications, making it crucial for both beginners and experienced coders.