Epic Python Coding

Interactive Coding Adventures

For Kids

By Mike Gold

Epic Python Coding: Interactive Coding Adventures for Kids

Mike Gold

This book is for sale at
http://leanpub.com/pythonplaygroundinteractivecodingadventuresforkids

This version was published on 2024-01-02

* * * * *

* * * * *

Dedicated to my son, who likes to learn, but won't admit it.

Table of Contents

Epic Python Coding: Interactive Coding Adventures for Kids

Figure 1. Python Programmer

Introduction to “Epic Python Coding”

Welcome to the Amazing World of Python Programming!

Hey there, future programmer! Are you ready to embark on an exciting adventure into the world of Python programming? If you love solving puzzles, creating stories, or just learning super cool new things, then you are in the right place!

What is Python? No, we're not talking about snakes here! Python is a special tool that computer experts use to tell computers what to do. Just like you use words to tell a story, programmers use a language called Python to give instructions to computers. And the best part? Python is easy and fun to learn!

Why Python? Python is like the friendly, easy-to-understand language of the computer world. It's perfect for beginners like you. It's used all over the world to do amazing things like building video games, solving math problems, drawing pictures, and even helping scientists learn about space!

In this book, we're going to explore Python together. You'll learn how to make the computer do cool things like:

- Solve math puzzles
- Play with words and sentences
- Draw colorful shapes and patterns
- And even create your very own computer game!

You don't need to be a computer genius to start. All you need is your curiosity and excitement to learn. So, grab your explorer's hat and let's dive into the wonderful world of Python programming. Your adventure is just about to begin, and who knows what amazing things you'll create!

Are you ready? Let's go!

Chapter 1: Setting Up Your Adventure

Installing Python

Welcome to your first step in becoming a young Python programmer! Before we start creating cool stuff, we need to set up Python on your computer. Don’t worry, it’s easy, and I’ll guide you through each step!

1. Visit the Python Website:

- Ask an adult to help you open your web browser (like Chrome, Firefox, or Safari).
- Type in the website address: www.python.org.
- You’ll see a screen like this:

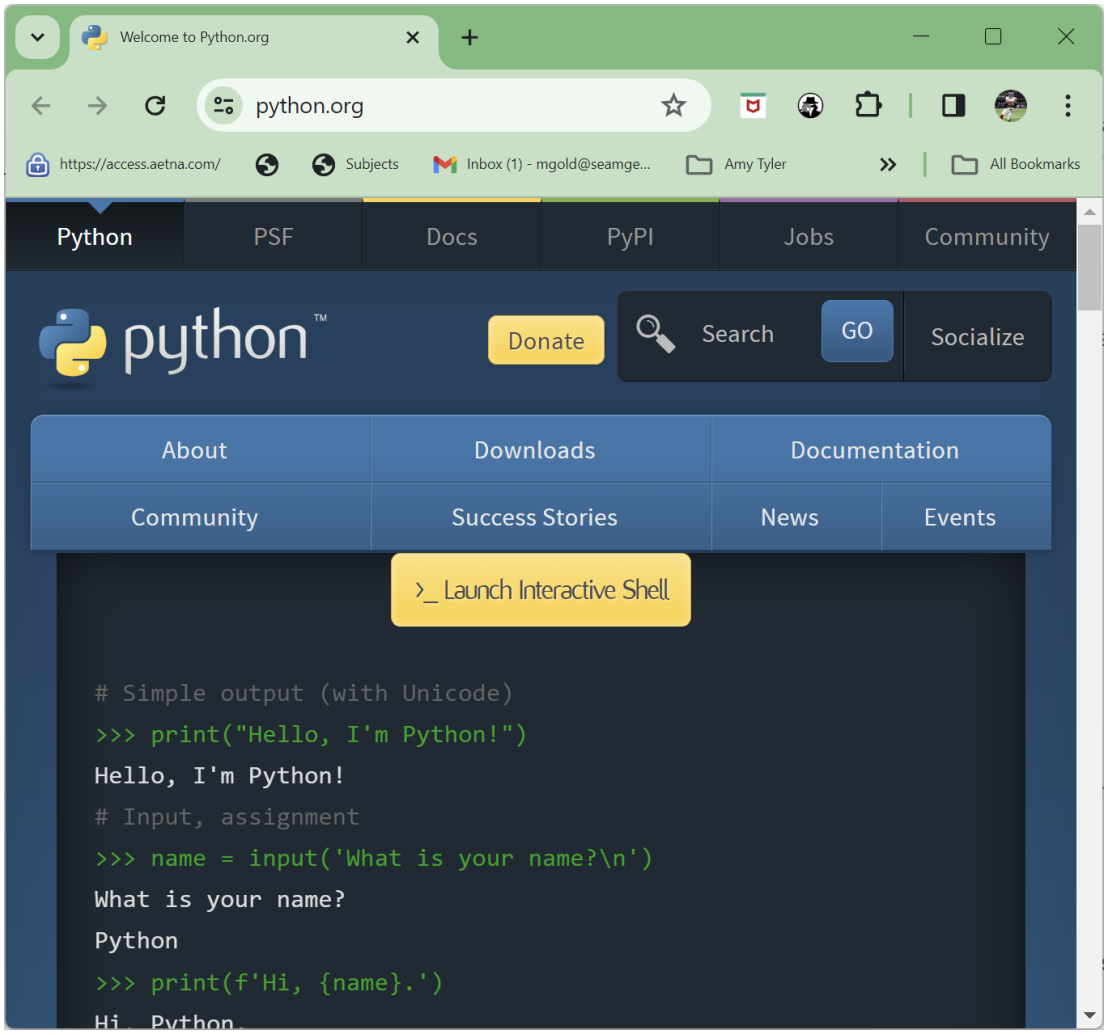

Figure 2. Website Screenshot of Python.org

2. Download Python:

- Click on the “Downloads” tab.

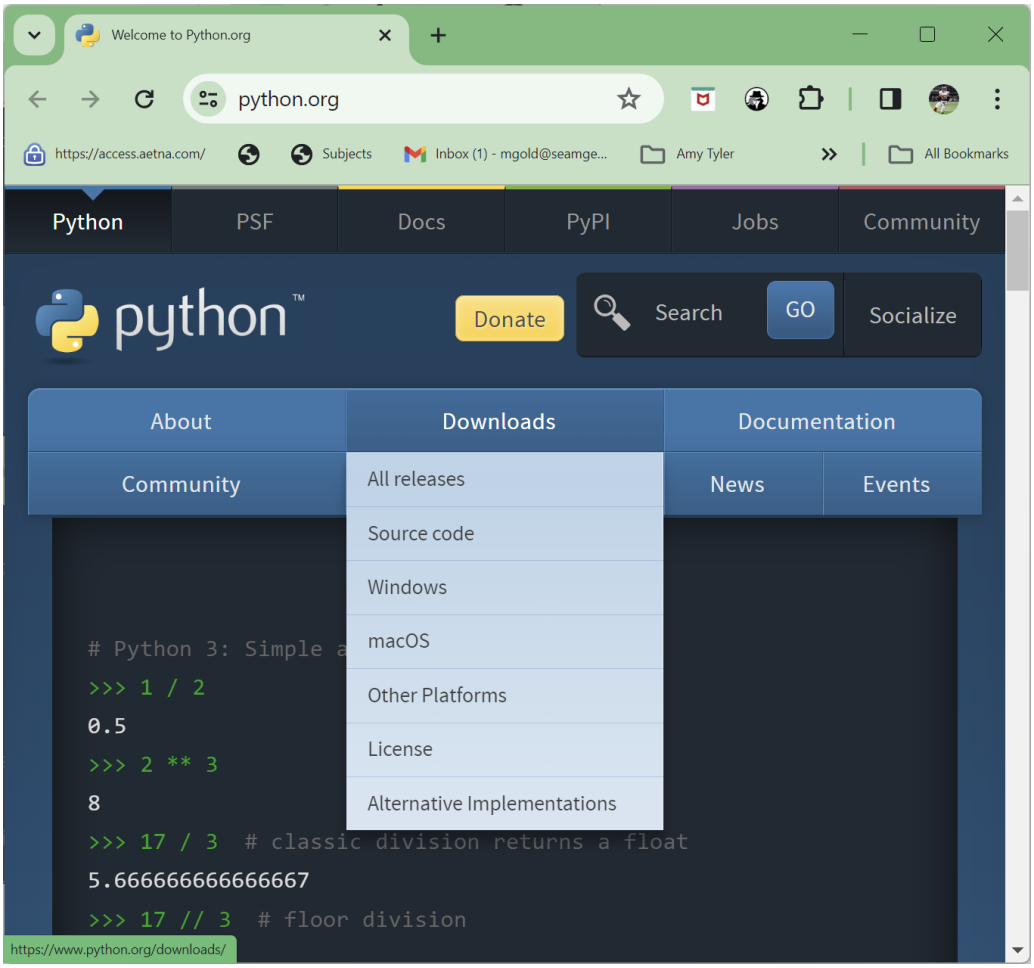

Figure 3. Download menu

- Python knows what kind of computer you have, so it will suggest the best version for you. It will look something like this:

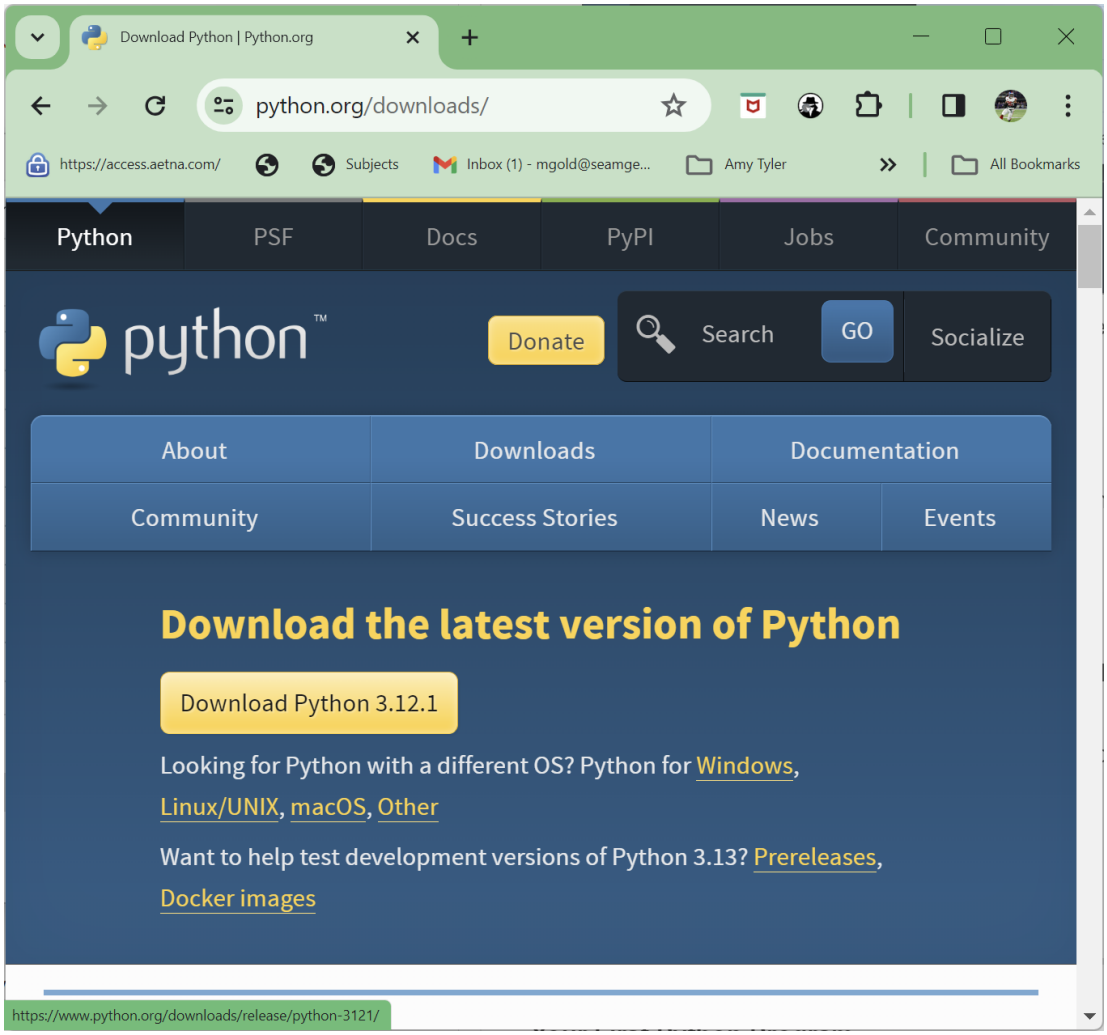

Figure 4. Download Button on Python.org

- Click the download button. A file will start downloading. Once it's done, click on it to open it.

3. Install Python:

- A window will pop up to start the installation. Make sure to check the box that says "Add Python to PATH" – this is super important!
- Then, click "Install Now" and wait for the magic to happen!

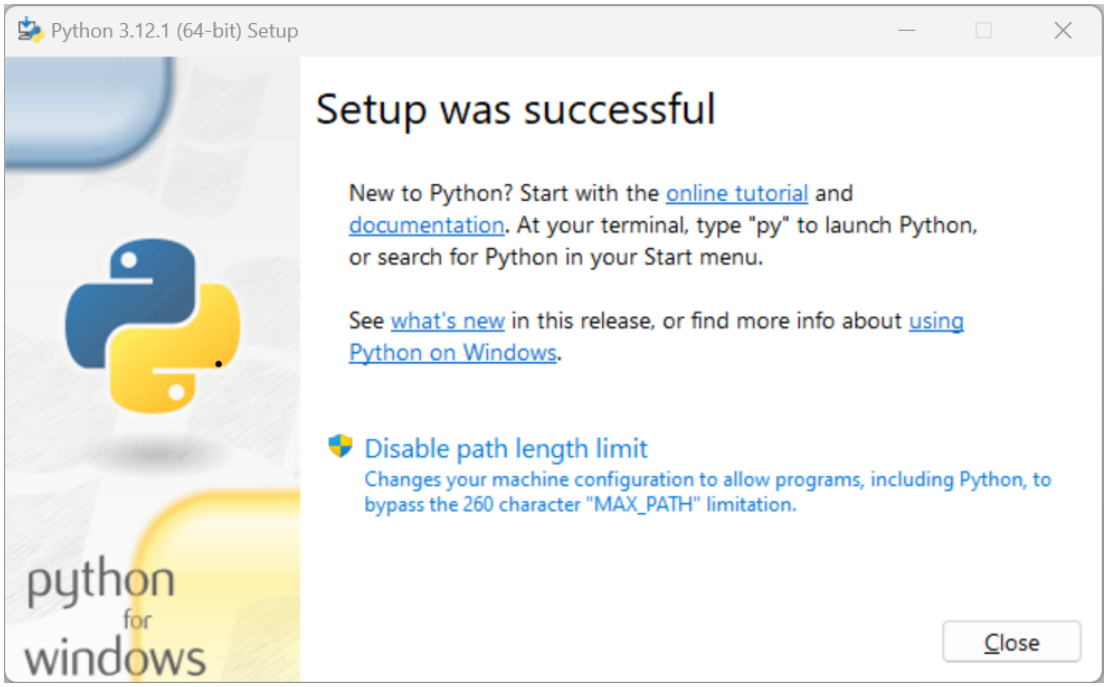

Figure 5. Python Installation Window

Congratulations! You've just installed Python!

Your First Python Program

Now, let's write your very first Python program. We're going to make the computer display a message. It's like teaching your computer to talk!

1. Open Python:

- Search for Python in your computer's search bar.
- Click on the Python application – it's usually called "IDLE" (Python's editor).

2. Write Your Program:

- A window will open where you can type your code.
- Type exactly this: `print("Hello, Python World!")`

3. Run Your Program:

- Click on “Run” in the menu, then select “Run Module”.
- Look! The computer says hello back to you in the output window!

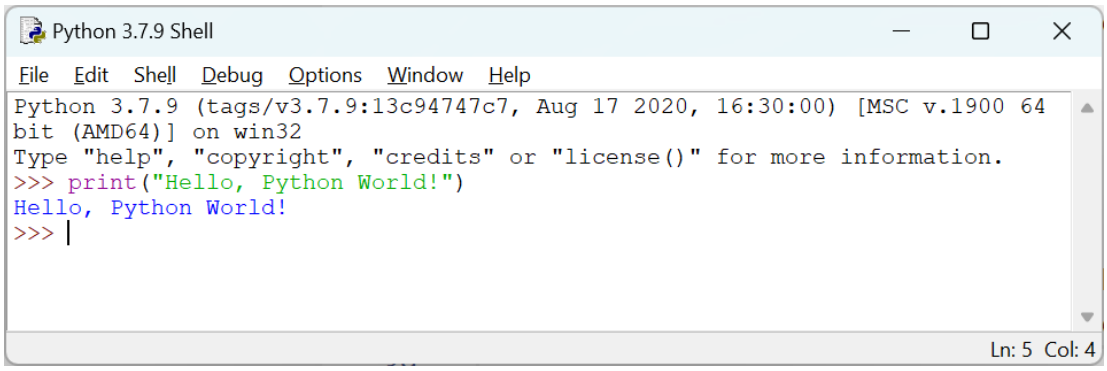

Figure 6. Python IDLE with Hello World Program

You just wrote your first program!

Understanding the Python Shell

The place where you wrote your program is called the Python Shell. It’s like a playground for your code. You can write instructions here, and Python will respond. Try typing `print("I'm a Python programmer!")` in the shell and see what happens!

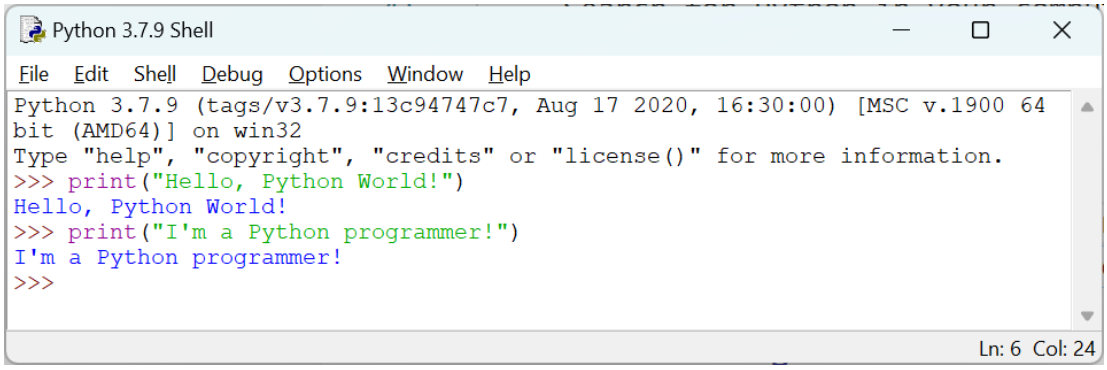

Figure 7. I’m a Python Programmer

In this chapter, we learned how to set up Python and wrote our first program. You’re now on your way to becoming a great Python programmer. Keep up the great work!

Chapter 2: Discovering Python Basics

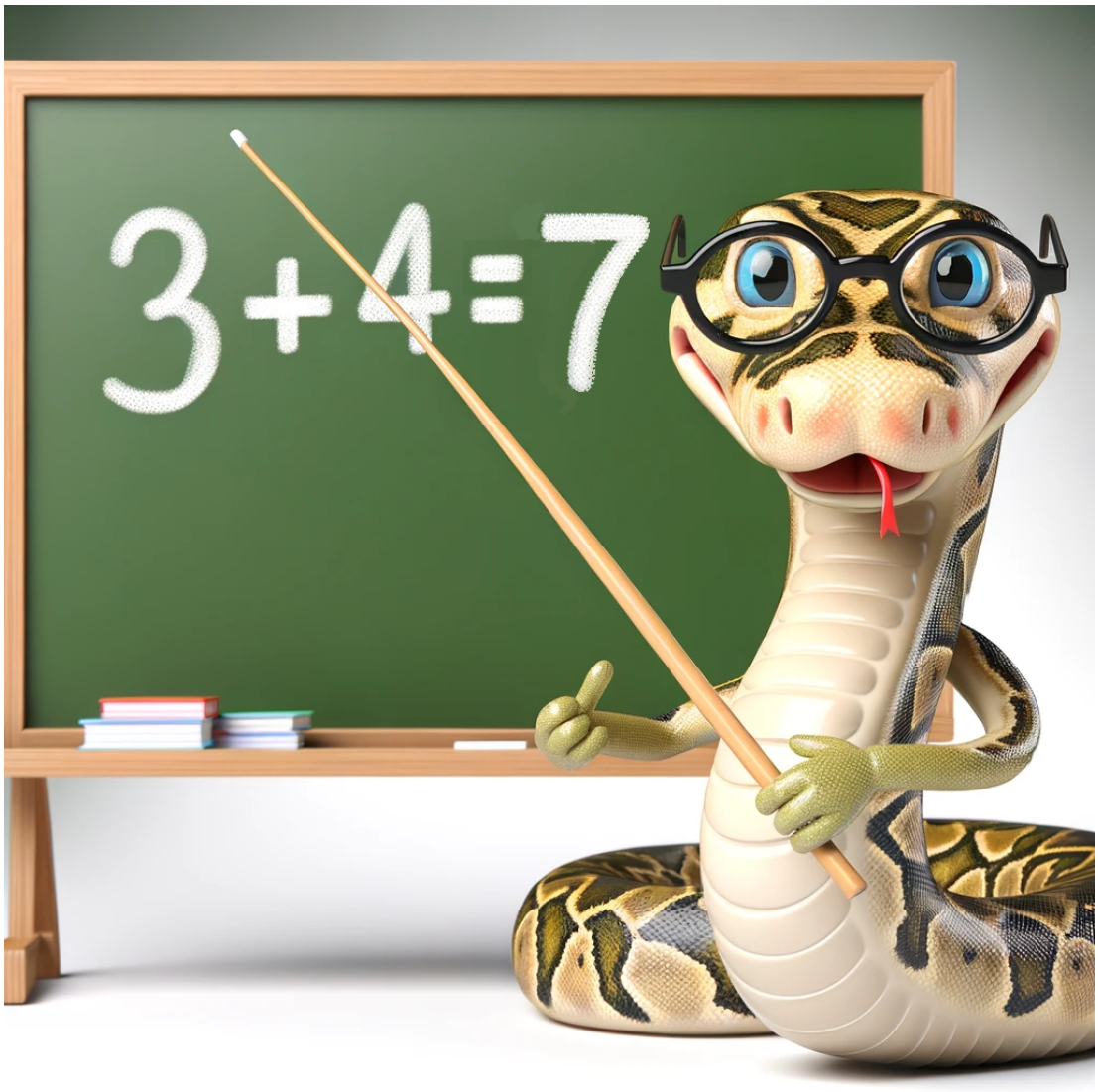

Figure 8

Playing with Numbers

Welcome to the world of numbers in Python! Did you know that Python is really good at math? Let’s play with some numbers and see what we can do.

1. Simple Math Operations:

- Open Python's IDLE, just like we did before.
- Let's start with something simple. Type `3 + 4` and press Enter.

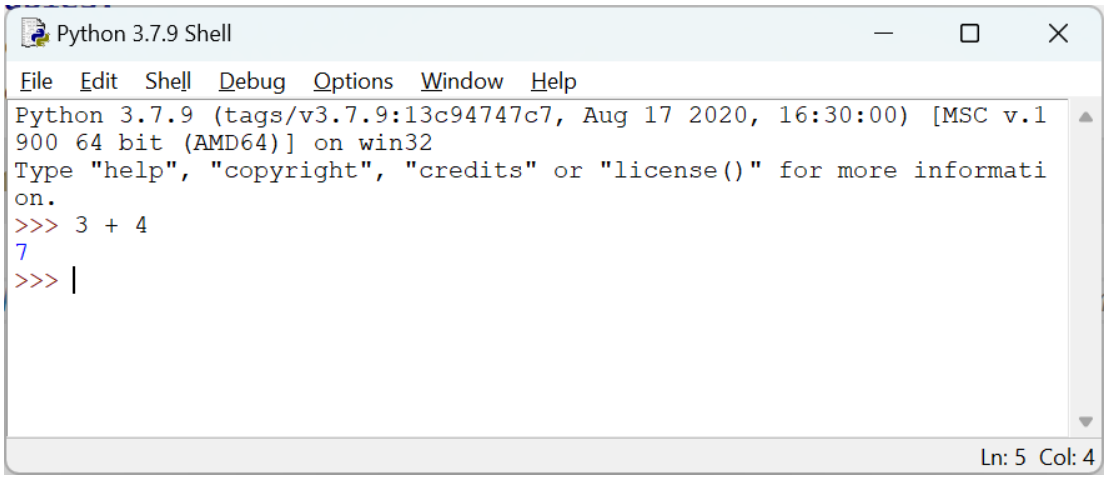

Figure 9. Adding in Python

- Wow, Python can add!
- You can try other operations too:
  - For subtraction, use `-` (like `5 - 2`).
  - For multiplication, use `*` (like `3 * 4`).
  - For division, use `/` (like `10 / 2`).

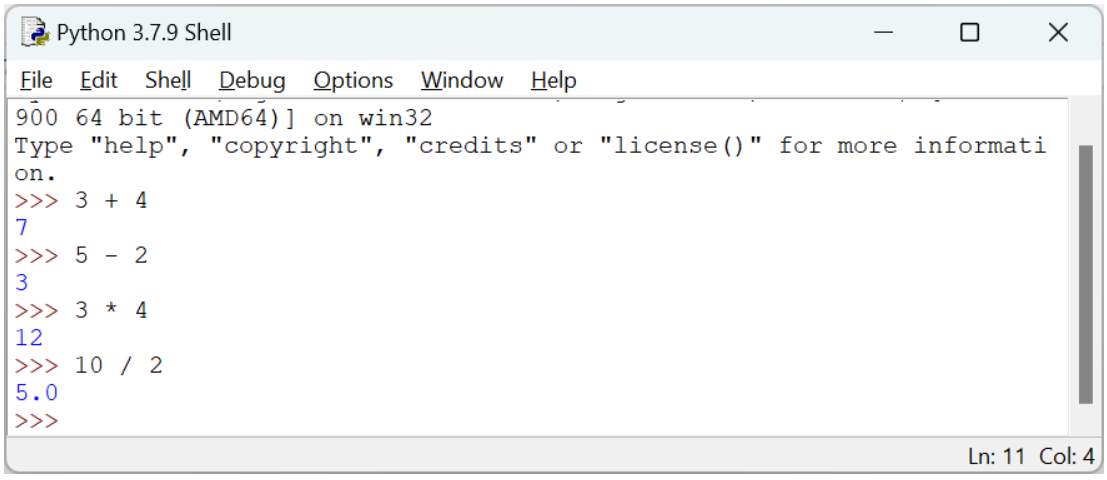

Figure 10. Other Math Functions in Python

2. Introducing Variables:

- Think of variables like little boxes where you can store things. In Python, you can store numbers in these boxes.

Figure 11. variable number in a box

- Let’s create a variable. Type `my_age = 10` and press Enter. You just told Python that your age is 10!
- Now, type `my_age` and press Enter. Python remembers it’s 10!

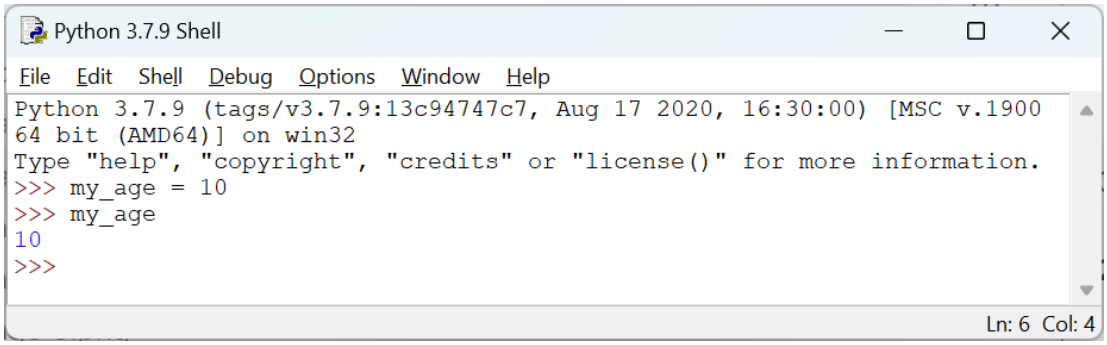

Figure 12. Storying my age in a variable

Doing Math with Variables

Great job learning about numbers and variables! Now, let's see how we can use variables to do math. Remember `my_age`? Let's use it in some math operations.

1. Adding to a Variable:

- First, let's remind Python how old you are. Type `my_age = 10` and hit enter.

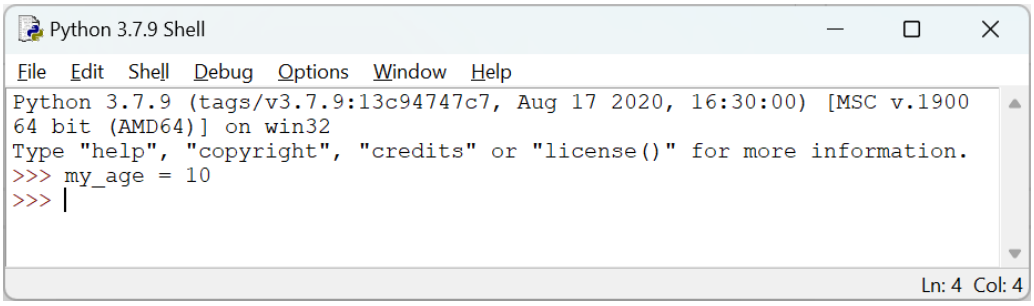

Figure 13. Setting my_age variable

- What if we want to know how old you'll be in 3 years? We can add 3 to your age.

- Type `future_age = my_age + 3`. This tells Python to add 3 to your current age and save the answer in a new variable called `future_age`.

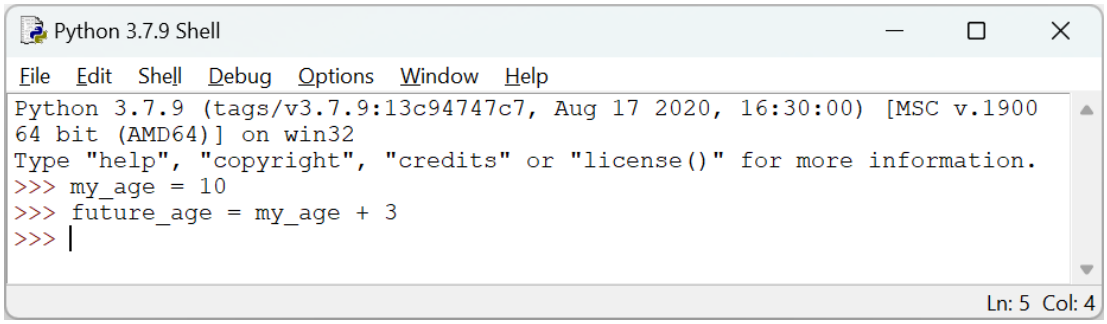

Figure 14. Creating a future age variable

- Now, type `print(future_age)`. Python will show you how old you'll be in 3 years!

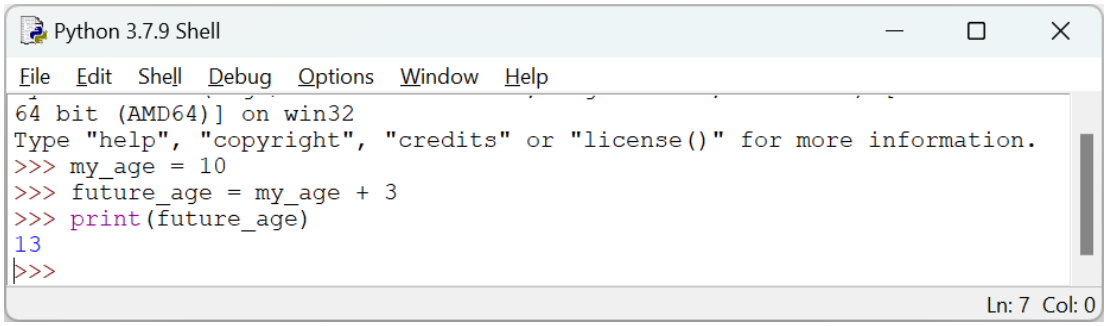

Figure 15. Printing the future age variable

2. More Math with Variables:

- You can do all kinds of math with variables. Let's try more.
- Half your age: Type `half_my_age = my_age / 2` and then `print(half_my_age)`.

```
>>> my_age = 10
>>> half_my_age = my_age/2
>>> print(half_my_age)
5.0
```

Figure 16. Compute half my age

- Double your age: Type `double_my_age = my_age * 2` and then `print(double_my_age)`.

```
>>> double_my_age = my_age * 2
>>> print(double_my_age)
20
```

Figure 17. Double my age

- Subtracting years: Type `younger_age = my_age - 2` and then `print(younger_age)`.

```
>>> younger_age = my_age - 2
>>> print(younger_age)
8
```

Figure 18. Younger age

3. Changing the Variable:

- Variables can change. Remember a variable is like a box, you can take the old number out of the box and put a new number in it! If you have a birthday and you're now 11, just update it!

- Type `my_age = 11` and hit the enter key. Now `my_age` has a new value.

```
>>> my_age = 11
>>> print(my_age)
11
```

Figure 19. Changing my_age to 11

- If you do `print(future_age)` again, it still shows your future age as if you were 10.

```
>>> print(future_age)
13
```

Figure 20. Future age not updated

That's because `future_age` was calculated with your old age. To update it, you need to calculate it again with your new age.

```
>>> future_age = my_age + 3
>>> print(future_age)
14
```

Figure 21. Future age recalculated

Look at you, doing math with Python like a pro! Variables are super handy in programming, and you're using them like a champ. Keep practicing, and soon you'll be able to solve all sorts of cool problems with Python!

Python Example: Calculating and Sharing Your Future Age in One Go

What Are We Doing? We're going to tell Python how old you are and then ask Python to calculate hand print how old you'll be in 5 years. We will do this all in just two lines of code!

Step-by-Step:

1. **First, We Tell Python Your Age:**
    - To Review: When you type `my_age = 10`, you're like saying to Python, "Hey, my age is 10 years old!" Think of `my_age` as a box where you keep your age.

2. **Next, We Do Some Math and Print at the Same Time:**
    - Now, the fun part! When you type `print("In five years, I will be", my_age + 5)`, you're asking Python to do two things:
        - **Add to Your Age:** Python takes the number in your `my_age` box (which is 10) and adds 5 to it. It's like saying, "What's 10 plus 5?"
        - **Print a Message:** The `print` part tells Python to write out a message. The message starts with the words `"In five years, I will be"`. Then, Python adds the answer to "10 plus 5" right after those words.

What Happens When You Run the Code:

- Python will think, "Okay, 10 plus 5 is 15," and then it will show on the screen: `In five years, I will be 15.`

```
>>> my_age = 10
>>> print("In five years, I will be", my_age + 5)
In five years, I will be 15
```

Figure 22. Alt text

Why It's Cool:

- You just told Python to remember your age and do math with it! And Python did all of that and even wrote a sentence for you. That's pretty awesome, right?

What if You Get an Error?

Common Mistake: Forgetting to Define a variable

- Suppose you accidentally typed `print("In five years, I will be", age + 5)` instead of using `my_age`.

Python's Response:

- Python will say something like: `NameError: name 'age' is not defined`

```
>>> future_age = age + 3
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    future_age = age + 3
NameError: name 'age' is not defined
```

Figure 23. Error in code

Why Did This Happen?

- This error means Python is confused because it doesn't know what `age` is. You told Python about `my_age`, but not `age`.

How to Fix It:

1. **Read the Error Message:**
   - It tells you that `age` is not defined.
2. **Check Your Code:**
   - Look at your code and compare it to the instructions. Do you see the difference?
3. **Make Corrections:**
   - Change `age` to `my_age` in your print statement.
4. **Try Again:**
   - Run your code once more.

Learning from Mistakes:

- Now you know how to fix a `NameError!` Remember, it's okay to make mistakes. Fixing them is how you learn to be a great programmer!

Creating a Story with Words

Now let's play with words, or as programmers call them, "strings."

1. String Basics:

- Strings are just groups of letters in Python. But, you have to tell Python it's a string by using quotes.
- Try typing `print("I love Python!").` See how Python shows your message?

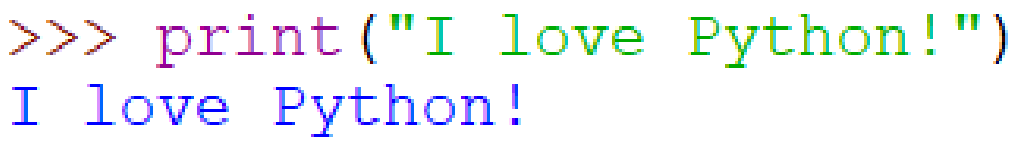

Figure 24. Print Python message

2. Combining Words and Sentences:

- You can make Python talk even more. Let's combine two strings.
- Type `greeting = "Hello" + " there!"` and press Enter.
- Now type `print(greeting).` Python just combined the words to say "Hello there!"

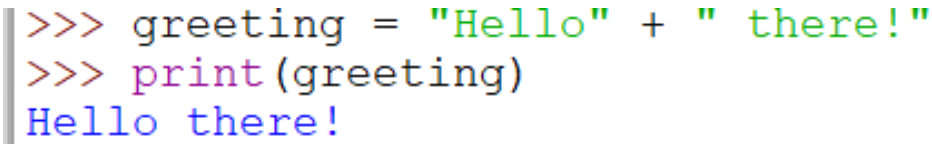

Figure 25. Combining strings

You're doing great! Now you know how to do basic math and create stories with words in Python. Keep practicing, and you'll be a Python pro in no time!

Chapter 3: Fun with Lists and Loops

Figure 26. Python Train

Making Lists

Welcome to a super fun part of Python – Lists and Loops! Let’s start with lists.

What are Lists?

- Think of a list in Python like your backpack where you can keep lots of different things. Just like you might have a list of your favorite toys, Python can keep a list of things you tell it, like numbers, words, or even other lists!

How to Create and Use Them:

1. **Creating a List:**
    - Open Python's IDLE.
    - Let's make a list of your favorite fruits. Type this:

Figure 27

```
1 my_fruits = ["apple", "banana", "cherry"]
```

    - You just told Python, "Here's a list of my favorite fruits!"
2. **Using the List:**
    - Want to see your list again? Just type `print(my_fruits)` and press Enter.
    - Python will show: `['apple', 'banana', 'cherry']`
    - Each item in your list is kept in order, like they are in your backpack.

```
>>> my_fruits = ["apple", "banana", "cherry"]
>>> print(my_fruits)
['apple', 'banana', 'cherry']
```

Figure 28. List of Fruits

Looping Around

Now, let's have some fun with loops! A loop is like a superpower in Python that lets you do something over and over again without getting tired.

Introduction to For-Loops:

- A for-loop in Python goes through each item in your list, one by one, and does something with it.

1. Printing items in a List with a For-Loop:

- Let’s go through each fruit in the list one at a time. Type this:

Figure 29

```
for fruit in my_fruits:
    print(fruit)
```

- This is like telling Python: “For every fruit in my list of fruits, say the name of the fruit.”

2. What Happens When You Run It:

- Python will print each fruit’s name, one by one:

Figure 30

```
apple
banana
cherry
```

- It’s like Python is taking each fruit out of your backpack and showing it to you.

3. Looping for a Number Game:

- Let’s play a game with numbers. First, make a list of numbers:

Figure 31

```
my_numbers = [1, 2, 3, 4, 5]
```

- Now, let’s add 10 to each number using a loop:

Figure 32

```
for number in my_numbers:
    print(number + 10)
```

- This tells Python: “For each number in my list, add 10 and then tell me the new number.”

4. See the Magic:

- Python will show you:

Figure 33

```
11
12
13
14
15
```

- Isn’t that cool? You just did math on a whole list of numbers, super fast!

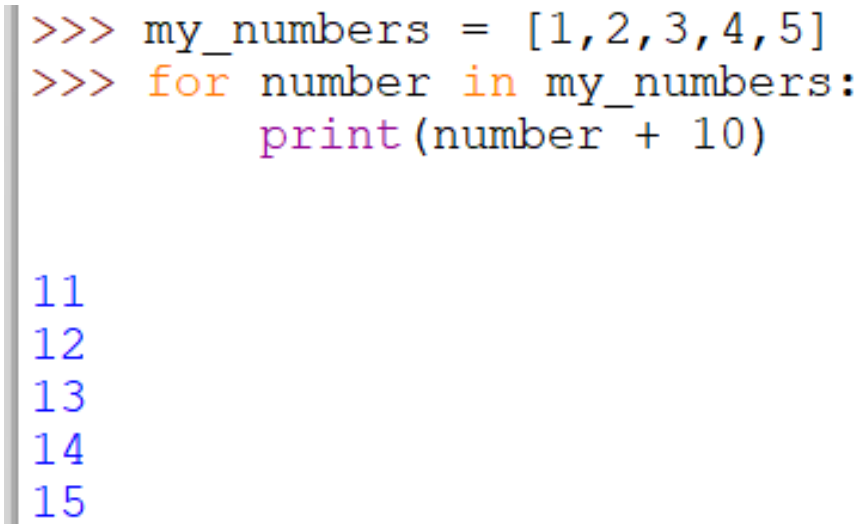

Figure 34. Adding 10 to numbers in a list

More Fun with Loops: Creating a Story!

Let’s try something really cool with loops. We can use a loop to take a bunch of words and put them together to make a sentence. It’s like building a train where each word is a train car!

Making a Sentence from Words:

1. **First, Make a List of Words:**
    - Let's make a list of words that will become our sentence. How about making a silly sentence? Type this:

      Figure 35

      ```
      1 silly_words = ["My", "dog", "ate", "my", "homework",
      2                "yesterday"]
      ```

    - You just created a list of words for our silly sentence!
2. **Using a Loop to Build the Sentence:**
    - Now, we'll use a loop to go through each word and add them together to make a sentence. Type this:

      Figure 36

      ```
      1 sentence = ""
      2 for word in silly_words:
      3     sentence = sentence + word + " "
      4 print(sentence)
      ```

    - Here's what you're telling Python: "For each word in my list of silly words, add it to my sentence, and put a space after it."
3. **Watch the Magic Happen:**
    - When you run this code, Python will show:

      Figure 37

      ```
      1 My dog ate my homework yesterday
      ```

    - You just made a whole sentence by looping through your words!

What Just Happened?

- You started with an empty sentence (`sentence = ""`).

- Then, the loop went through each word in your list (`for word in silly_words),` adding it to your sentence.
- The `" "` part added a space after each word, so they didn't all squish together.
- Finally, you used `print(sentence)` to see your silly sentence!

```
>>> silly_words = ["My", "dog", "ate", "my", "homework", "yesterday"]
>>> sentence = ""
>>> for word in silly_words:
        sentence = sentence + word + " "


>>> print(sentence)
My dog ate my homework yesterday
```

Figure 38. Building a sentence with loops

Isn't it amazing how you can take some words, loop through them, and make a whole sentence? You're becoming a real Python wizard! Keep practicing, and soon you'll be able to create all sorts of fun stories with your Python skills!

And that's how you make lists and use loops in Python! It's like having a magic wand to sort through your backpack, play number games, or make sentences. Keep practicing, and soon you'll be able to do even more magical things with Python!

Chapter 4: Making Decisions with Python

Figure 39

Starting Your Python Adventure with IDLE Files

Before getting started on decisions in Python, let’s talk about a cool tool called the IDLE Python file. It’s like your magic notebook for writing and saving your Python adventures!

Saving and Running Your Code in Python IDLE

So far, you been typing your Python adventures directly in IDLE's command line. That's a great start! But did you know you can also write your code in a Python file and save it to run later? It's like writing a story in a notebook and then reading it whenever you want!

Here's How You Do It:

1. **Open a New File:**
    - In Python's IDLE, go to `File` in the top menu and choose `New File`. This opens a new window where you can write your code, just like typing in a big, blank page.

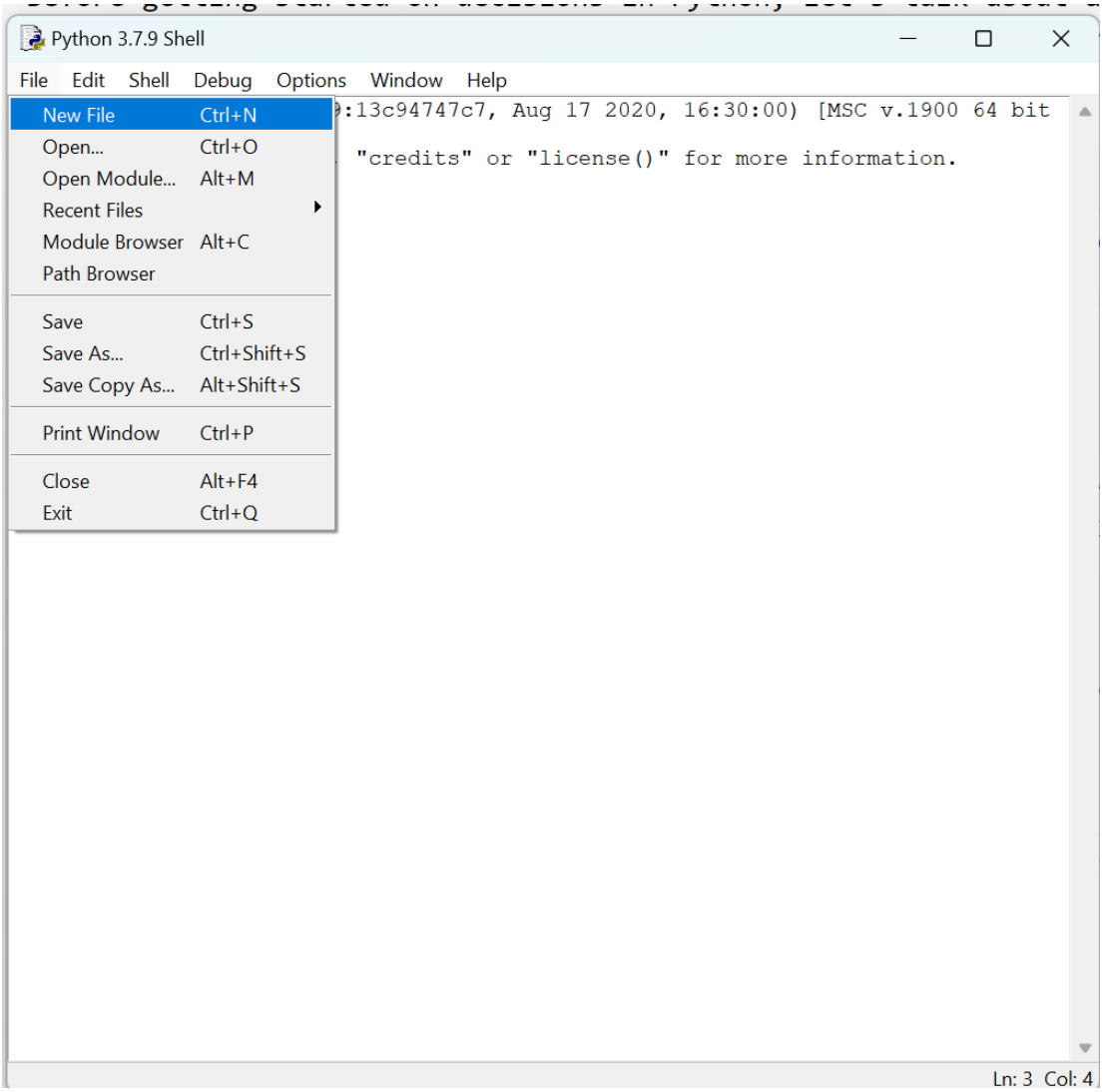

Figure 40. Opening a Python File

2. **Write Your Python Story:**

    - Type your code into this new window. You can write all sorts of things, like your variables, if-statements, loops, and lists.

3. **Save Your Adventure:**

    - To save your code, go to `File` and then `Save` or `Save As`. Choose a place on your computer to save it, like a special folder for your

Python adventures. Give it a name ending in `.py`, like `my_adventure.py`.

4. **Run Your Code:**
    - After saving, you can run your code to see it in action. Just click `Run` and then `Run Module`, or simply press `F5` on your keyboard. Watch as Python brings your written code to life in the IDLE shell window!

By saving your code in a file, you can come back to it anytime and make changes or try new things. It's like having a diary of all your coding adventures that you can read and add to whenever you want!

If This, Then That

Get ready for some fun because we're going to learn how to make decisions in Python! Just like you decide every day what to wear or what to eat, we can teach Python to make decisions using something called "if-statements."

Understanding If-Statements:

- An "if-statement" is Python's way of choosing between different options. It's like saying, "If this is true, then do that; otherwise, do something else."

1. Your First If-Statement:

- Let's start with something simple. What if we want Python to say "Good morning" if it's morning and "Good night" if it's night? We can do that with an if-statement!
- First, we'll decide it's morning. Type the following into your IDLE file and save the file in a folder. :

Figure 41

```
1 is_morning = True
2 if is_morning:
3     print("Good morning!")
```

- Here, `is_morning = True` is like telling Python, “It’s morning time.” And the if-statement says, “If it’s morning (`if is_morning`), then say ‘Good morning!’”

2. Run Your Code:

- Run the code by either hitting the F5 key on your keyboard or by choosing Run —> Run Module from the menu. When you run this code, Python sees that `is_morning` is True, so it follows the instruction and says `Good morning!`

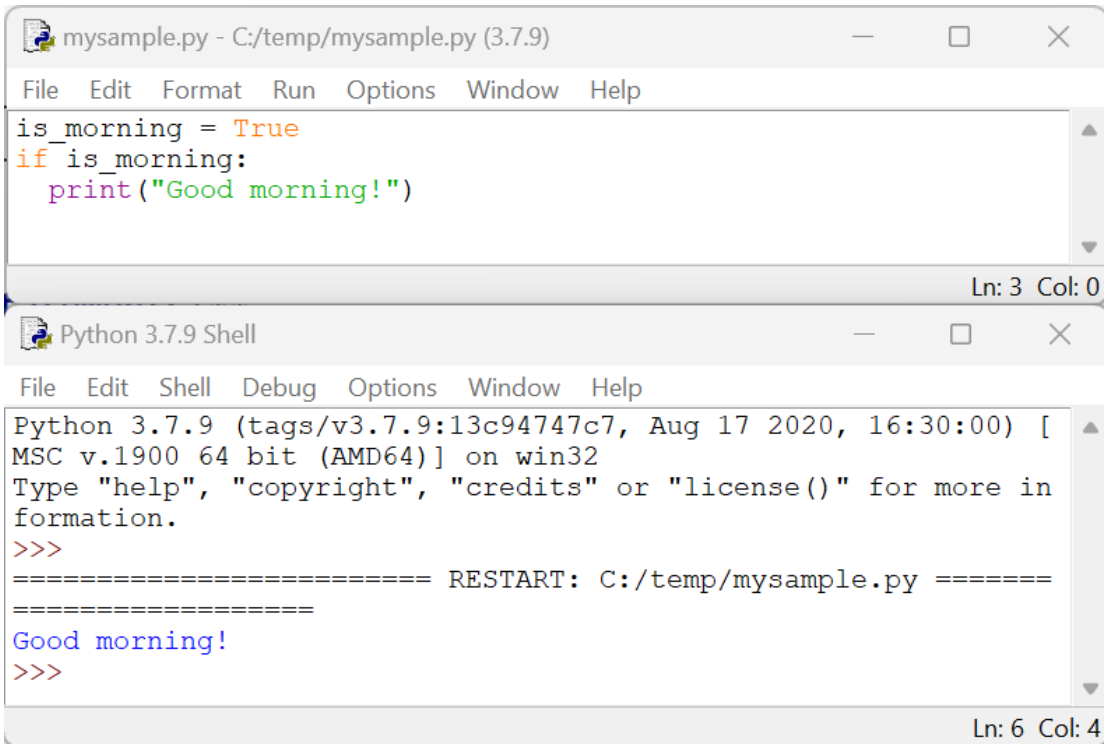

Figure 42. Running the decision code in a IDLE File

3. Adding More Choices:

- But what if it’s not morning? We can add another part, called an “else” part, to tell Python what to do if it’s not morning.
- Let’s add to our code:

Figure 43

```
1 is_morning = False
2 if is_morning:
3     print("Good morning!")
4 else:
5     print("Good night!")
```

- Now, because `is_morning` is False, Python will skip the "Good morning" part and go straight to the else part, saying `Good night!`

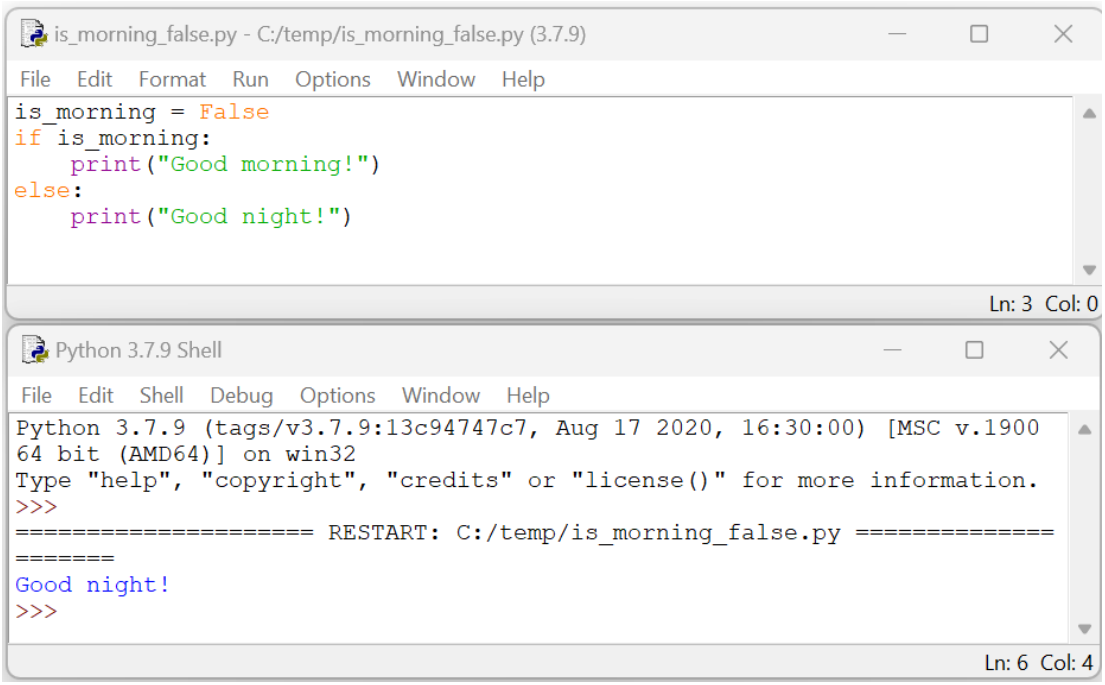

Figure 44. Else case in greeting

Simple Choice-Based Program:

1. **Creating a Small Game:**
   - Let's make a mini-game! We'll ask Python to decide if a number is big or small.
   - Type this game code:

Figure 45

```
1 number = 5
2 if number > 10:
```

```
    print("That's a big number!")
else:
    print("That's a small number!")
```

- Here, Python checks if the number is greater than 10. If it is, it says it's big. If not, it says it's small.

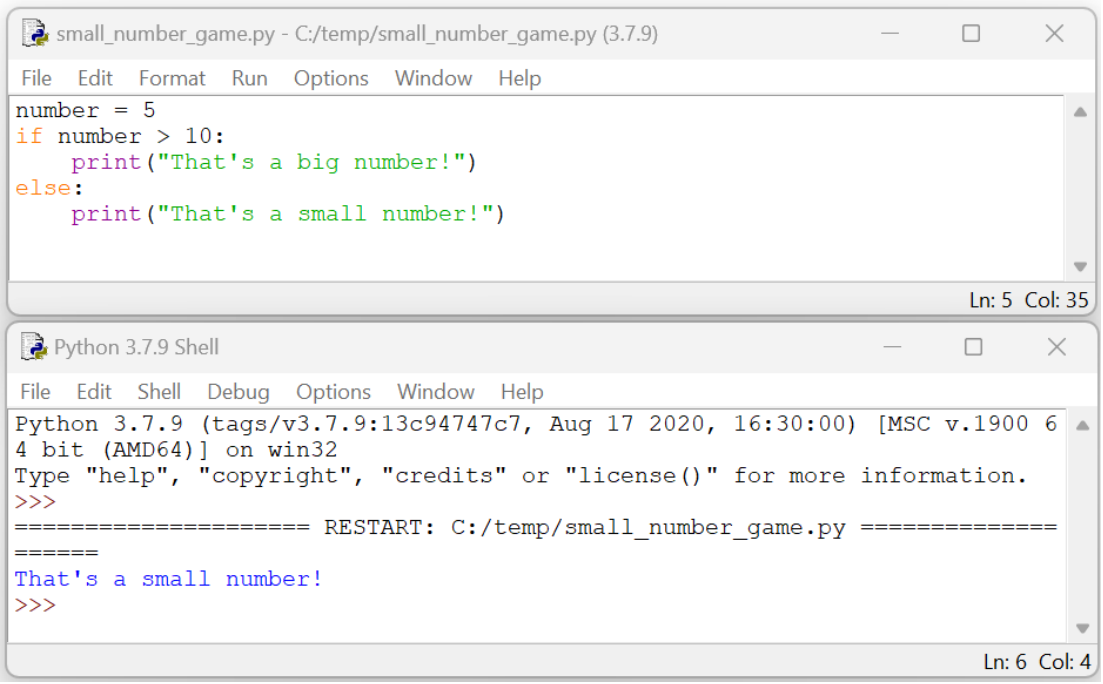

Figure 46. Checking if a number is big or small

2. **Try Changing the Number:**
   - Change `number = 5` to different numbers like 12, 8, or 20 in your IDLE file, and see what Python says each time.

You just learned how to make decisions with Python! Isn't it cool how you can tell Python to choose between different things? Remember, programming is all about giving instructions and making decisions. Keep practicing, and you'll be a decision-making expert in no time!

Beware of the Sneaky Indentation Errors in Python!

Hey awesome coder! Now that you're saving your Python adventures in a file, there's something tricky called **indentation errors** you should know about. Don't worry, they're easy to handle once you know what to look for!

What is an Indentation Error?

- In Python, the spaces at the beginning of a line of code are super important. They're like the secret code that tells Python which instructions belong together. This is called "indentation." If Python gets confused about the spaces, it will give you an "Indentation Error."

Example of an Indentation Error:

Let's look at your `if` statement code again about deciding if a number is big or small:

Figure 47

```
number = 5
if number > 10:
    print("That's a big number!")
else:
    print("That's a small number!")
```

This code works perfectly because the spaces at the beginning of the lines under `if` and `else` are lined up just right. But what if they weren't? Let's make a sneaky indentation error and see what happens:

Figure 48

```
number = 5
if number > 10:
print("That's a big number!")
else:
    print("That's a small number!")
```

Can You Spot the Mistake?

- The line `print("That's a big number!")` should have spaces at the start, just like the `print` line under `else`. But it doesn't!

What Does Python Say?

- Python will see this and get confused. It will give you an error message like this:

Figure 49

```
IndentationError: expected an indented block
```

How to Fix It:

- Just add spaces (or press `Tab`) at the beginning of the `print("That's a big number!")` line so it lines up with the `print` under `else:`

Figure 50

```
number = 5
if number > 10:
    print("That's a big number!")
else:
    print("That's a small number!")
```

Now It's Perfect!

- With the spaces all lined up, Python understands your code and can decide if the number is big or small without any trouble!

Remember, keeping your lines neat and in line is key to keeping Python happy!

How to Avoid Indentation Errors:

1. **Keep Your Spaces Consistent:**
    - When you write code under things like `if` statements or `for` loops, make sure to start each line in the same place. It's like

lining up your toys neatly in a row.

2. **Use the Tab Key:**
    - A handy trick is to press the `Tab` key on your keyboard to create spaces. This keeps your indentation nice and tidy.
3. **Check Your Code:**
    - If Python tells you there's an indentation error, don't fret! Just check your code to see if some lines start in different places and straighten them up.
4. **Ask for Help if You're Stuck:**
    - Sometimes finding these little errors can be like looking for hidden treasure. If you're stuck, ask someone to help you spot the misplaced spaces.

Remember: Keeping your code neat is like keeping your room tidy - it helps you find and fix problems faster, and it's a super important part of being a great programmer!

Happy coding and watch out for those sneaky indentation errors!

Chapter 5: Creative Colors and Shapes

)

Drawing with Python

Get ready for some colorful fun! Did you know that Python can be used to create art? Yes, with Python, you can draw shapes and color them in. We'll

use something called Turtle graphics. It’s like having a little turtle that draws on your screen as it moves around!

Using Turtle Graphics to Draw Shapes:

- Turtle graphics is a cool way in Python to make drawings. The ‘turtle’ is like a little artist that follows your commands to move and draw.

1. Setting Up Your Turtle:

- First, let’s get your turtle ready. Open a new file in Python’s IDLE, and type this at the top:

Figure 51

```
1 import turtle
2 my_turtle = turtle.Turtle()
```

- This tells Python, “I want to use the turtle, and let’s call it my_turtle.”

2. Moving Your Turtle to Draw:

- Now, let’s make your turtle draw a square. Add these lines to your code:

Figure 52

```
1 for _ in range(4):
2     my_turtle.forward(100)
3     my_turtle.right(90)
```

- This tells your turtle to move forward and then turn right, making each side of the square.

- Save your file with a `.py` at the end, like `my_square.py`.`Run` the code by hitting F5 and you’ll see the square being drawn by the turtle!

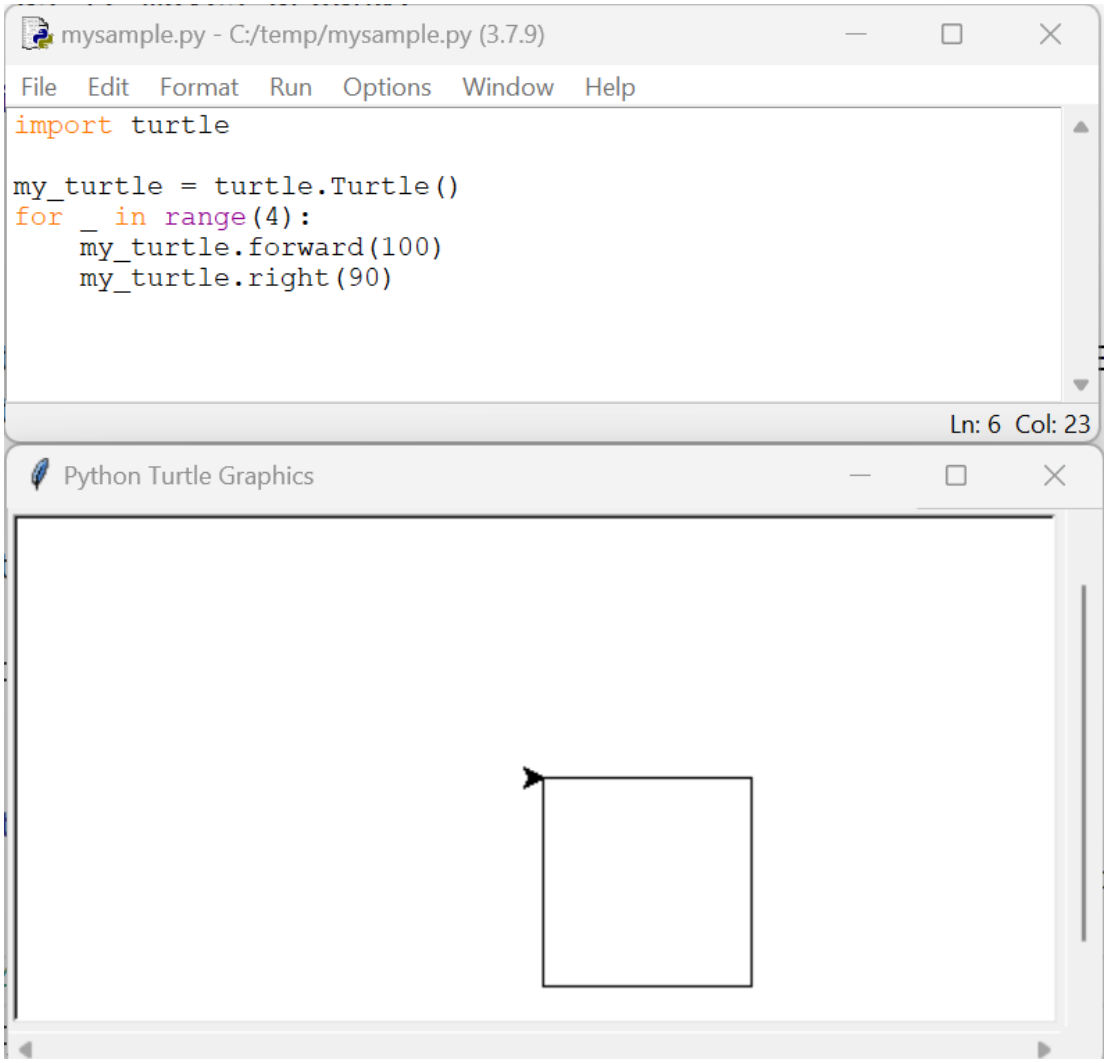

Figure 53. Drawing the Square

3. Adding Some Color:

- Let’s give your square some color. Before the loop, add this line:

Figure 54

```
1 my_turtle.fillcolor("blue")
2 my_turtle.begin_fill()
```

- And after the loop, add:

Figure 55

```
my_turtle.end_fill()
```

- Now your square will be filled with blue!

4. Running Your Code:

- Save your file again by hitting Save in the IDLE File menu`.
- Press `F5` to run it, and watch your turtle draw a blue square!

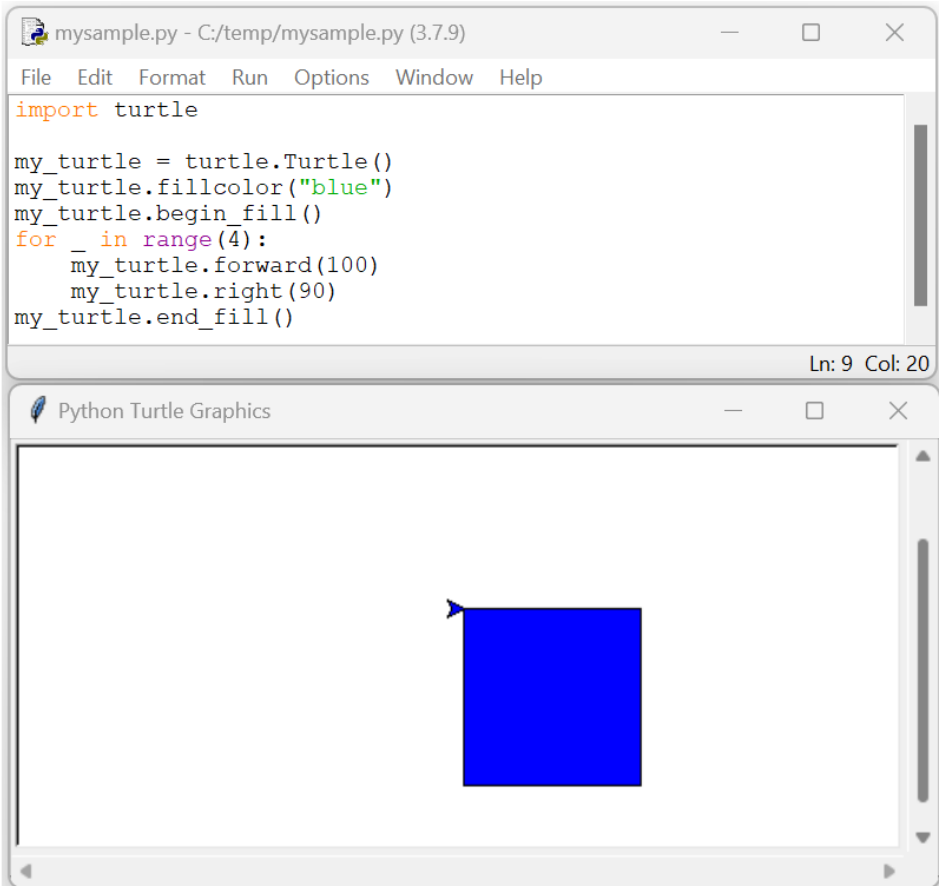

Figure 56. Turtle drawing blue square

Basic Color and Shape Commands:

- You can make your turtle draw different shapes and colors. Here is one idea:

- - Change `my_turtle.fillcolor("blue")` to another color like `"red"`, `"green"`, or `"yellow"`.
  - To draw a triangle, change the `range(4)` in the loop to `range(3)` and `my_turtle.right(90)` to `my_turtle.right(120)`.

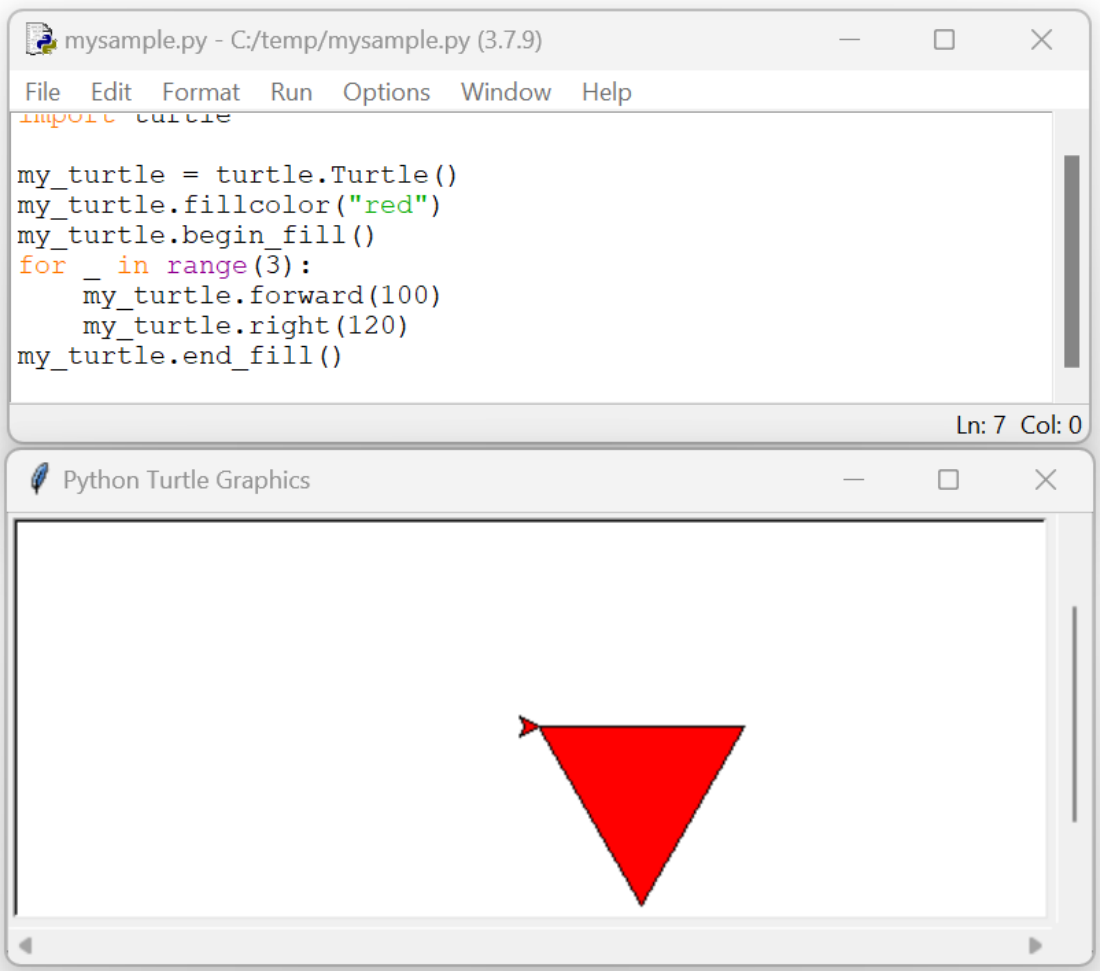

Figure 57. Drawing a red triangle

Taking Your Turtle Art to the Next Level

Wow, you did an awesome job drawing and coloring shapes with your Python turtle! Are you ready for something even more exciting? Let's make

your turtle draw a super cool spiral with lots of colors. This isn't just any spiral – it's a rainbow spiral that grows bigger and bigger!

Creating a Colorful Spiral: Just like you learned to draw squares and triangles, now we're going to use a loop to make a spiral. But this time, we'll add different colors to make it look like a rainbow. It's like telling your turtle to dance in a pattern while changing its colors!

By doing this, you'll see how changing a few simple instructions can create something totally new and amazing on your screen. Let's dive in and make your Python turtle create a beautiful, colorful spiral!

Steps to Create the Colorful Spiral:

1. **Set Up Your Turtle:**
    - Start with the basic setup for Turtle:

Figure 58

```
1 import turtle
2 my_turtle = turtle.Turtle()
3 my_turtle.speed(0)  # This makes your turtle draw fast
```

2. **Choose Your Colors:**
    - Let's pick some colors for the spiral. Add the following list of colors:

Figure 59

```
1 colors = ["red", "orange", "yellow", "green", "blue",
"pu\
2 rple"]
```

3. **Drawing the Spiral:**
    - Now, let's create the spiral with a loop. We'll make the turtle turn a little more each time and change colors. Add this code:

Figure 60

```
for i in range(50):
    my_turtle.color(colors[i % 6])  # Changes color each \
time
    my_turtle.forward(i * 5)       # Move forward by a gr\
owing distance
    my_turtle.right(60)             # Turn right by 60 de\
grees
```

4. Running Your Code:

- Save your file with a name like `colorful_spiral.py`.
- Press `F5` to run your code and watch as your turtle creates a beautiful spiral!

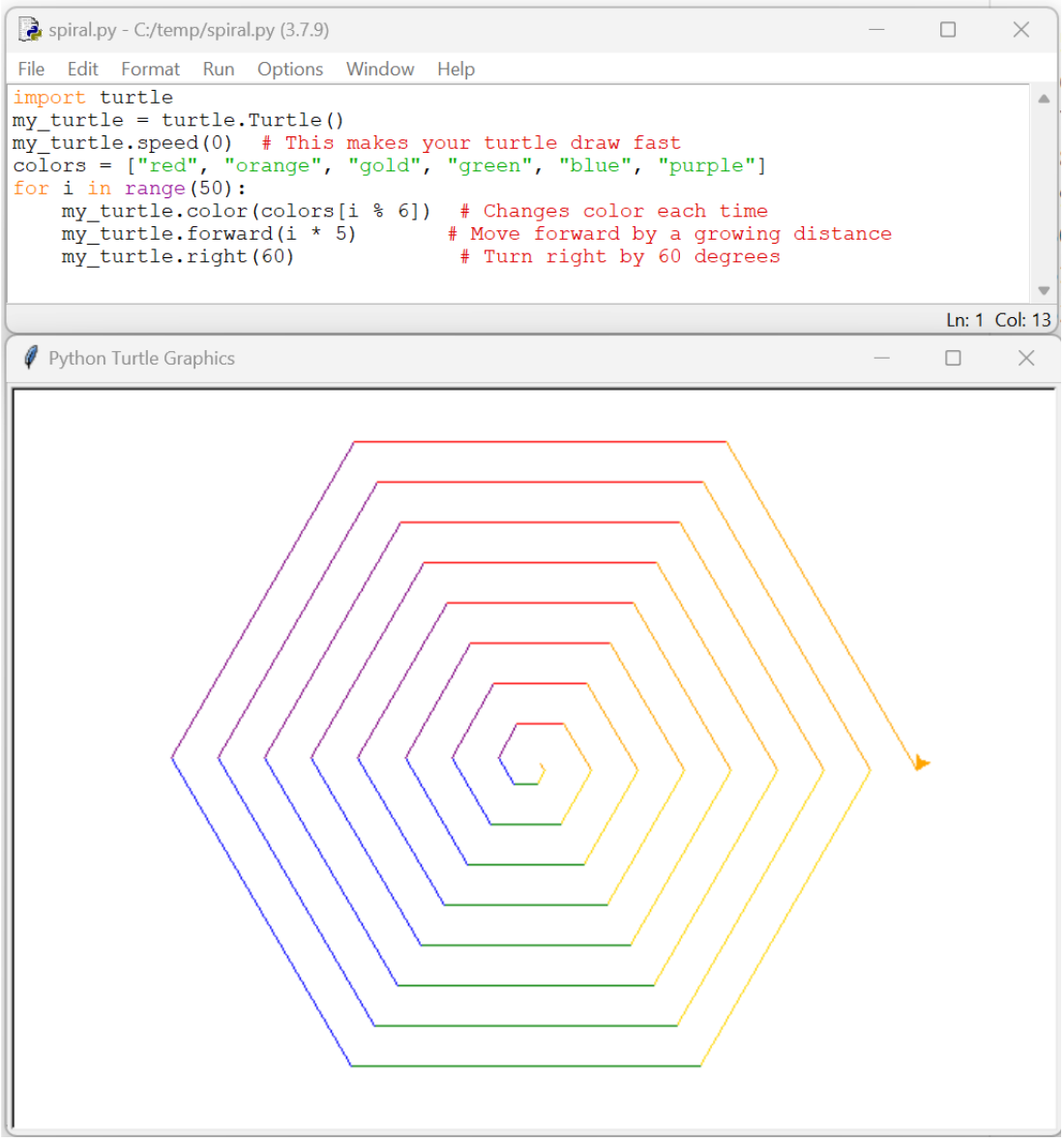

Figure 61. Drawing a spiral with turtle

What Happens in the Code:

- The `for i in range(50)` loop makes the code inside run 50 times.
- `my_turtle.color(colors[i % 6])` changes the color each time. The `% 6` makes sure the color index stays between 0 and 5 (since we have 6 colors).
- `my_turtle.forward(i * 10)` makes the turtle move forward. The `i * 10` makes each line longer than the last, creating the spiral effect.

- `my_turtle.right(60)` turns the turtle by 60 degrees to the right.

And that’s how you create amazing filled shapes and colorful spirals with Turtle in Python! Just like how a painter uses different brushes and colors to create a beautiful painting, you can use Turtle’s commands to draw your own digital art. Remember, you can choose any color you like for your shapes, and by repeating simple steps in a pattern, you create stunning spirals. Isn’t it fun to see how your computer can turn into a canvas for your imagination? Keep experimenting with different shapes and colors, and you’ll be a Turtle Python artist in no time!

Chapter 6: Python Magic Tricks: Fun with Functions

Figure 62

Creating Your Own Functions

Get ready for a new adventure in Python – learning about functions! Functions are like magic spells in coding. You create them once, and then

you can use them over and over again to do cool things. Let’s find out what functions are and how you can create your own!

What are Functions?

- Think of a function like a recipe in a cookbook. Each recipe tells you how to make a certain dish. In Python, a function is a set of instructions that tells the computer how to do something. Once you write a function, you can tell Python to follow those instructions whenever you want.

Why are Functions Useful?

- Functions are super useful because they help you avoid writing the same code again and again. If you find yourself doing the same thing multiple times, you can put that into a function and just call the function instead!

Writing Simple Functions:

1. **Your First Function:**
    - Let’s write a function that says hello. In your Python file, type this:

      Figure 63

      ```
      1 def say_hello():
      2     print("Hello, Python World!")
      ```

    - `def` is a special word that tells Python you’re making a new function. `say_hello` is the name of your function. The empty `()` means this function doesn’t need any extra information to work.
2. **Using Your Function:**
    - Just writing the function doesn’t make it do anything yet. You have to tell Python to use it. Underneath your function, type:

Figure 64

```
1 say_hello()
```

- This is like saying, “Hey Python, do the instructions I wrote in `say_hello`!”

3. **See What Happens:**

- Save your file and press `F5` to run it. You’ll see `Hello, Python World!` on the screen. Your function worked!

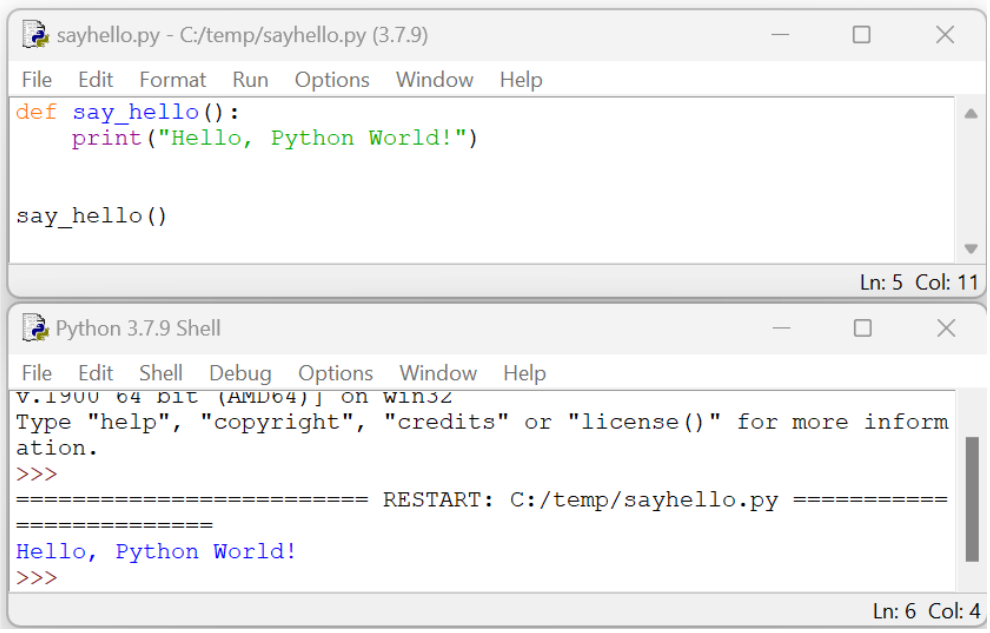

Figure 65. Your first function

4. **Creating a Counting Function:**

- Now, let’s make a function that counts to three. Type this new function:

Figure 66

```
1 def count_to_three():
2     for number in range(1, 4):
3         print(number)
```

- Then call your function by typing `count_to_three()` below it and run your code.

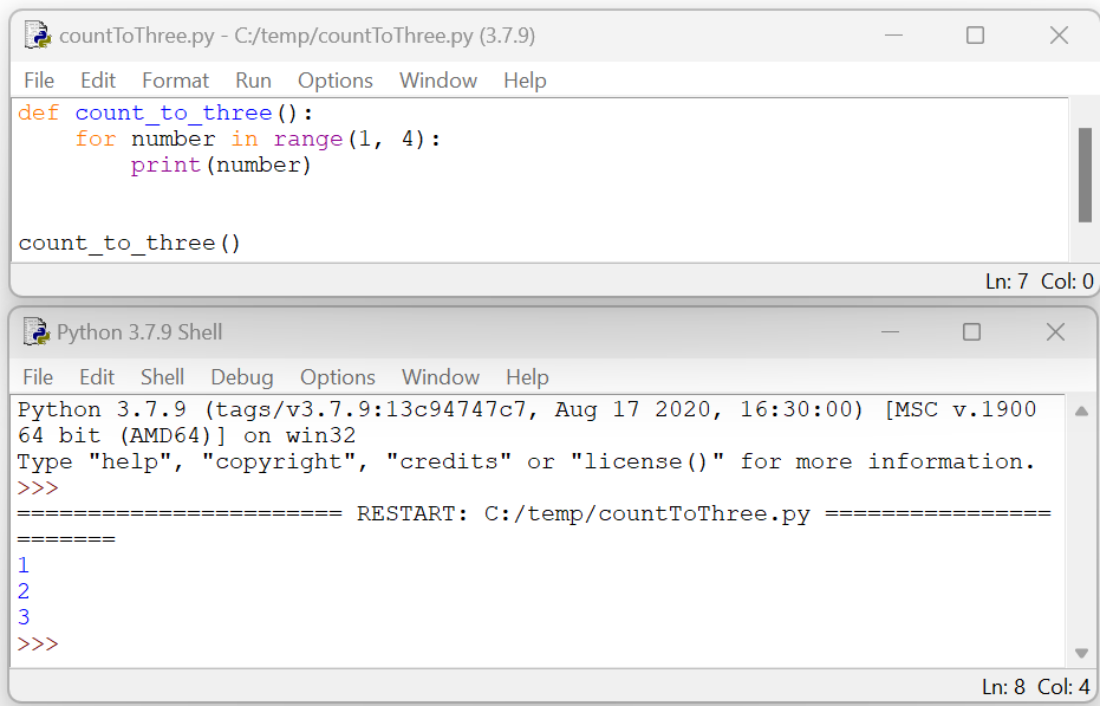

Figure 67. Count to three with a function

Let's talk about the `count_to_three` function we just created. It's a cool little piece of code that tells Python to count from 1 to 3. Let's break it down to see how it works!

The Recipe for Counting:

- Think of `count_to_three` as a recipe that tells Python how to count to three. Here's the code again:

Figure 68

```
1 def count_to_three():
2     for number in range(1, 4):
3         print(number)
```

What Each Part Does:

1. **Starting the Recipe (def count_to_three()):**
    - `def count_to_three():` is like the title of the recipe. It tells Python, “Here’s how you count to three.”
    - `def` means you’re defining a new function, and `count_to_three` is the name you’re giving it.
2. **The Counting Instructions (for number in range(1, 4)):**
    - Inside the function, we have a loop: `for number in range(1, 4).`
    - This is like saying, “For each number in the list of 1, 2, 3, do the following…”
    - Why 1 to 4, not 1 to 3? In Python, the range starts at the first number but stops just before the last number. So `range(1, 4)` includes 1, 2, and 3.
3. **Telling Python What to Do (print(number)):**
    - `print(number)` is the action part of our recipe. It tells Python to show each number on the screen.
    - So, Python will print 1, then 2, then 3, each on its own line.

Using the Function:

- Just writing the function isn’t enough. We have to tell Python to use it. That’s why we write `count_to_three()` after the function. It’s like saying, “Okay, Python, now follow the recipe to count to three.”

Seeing It in Action:

- When you run this code, Python reads the recipe and follows the steps, counting 1, 2, 3 out loud for you to see!

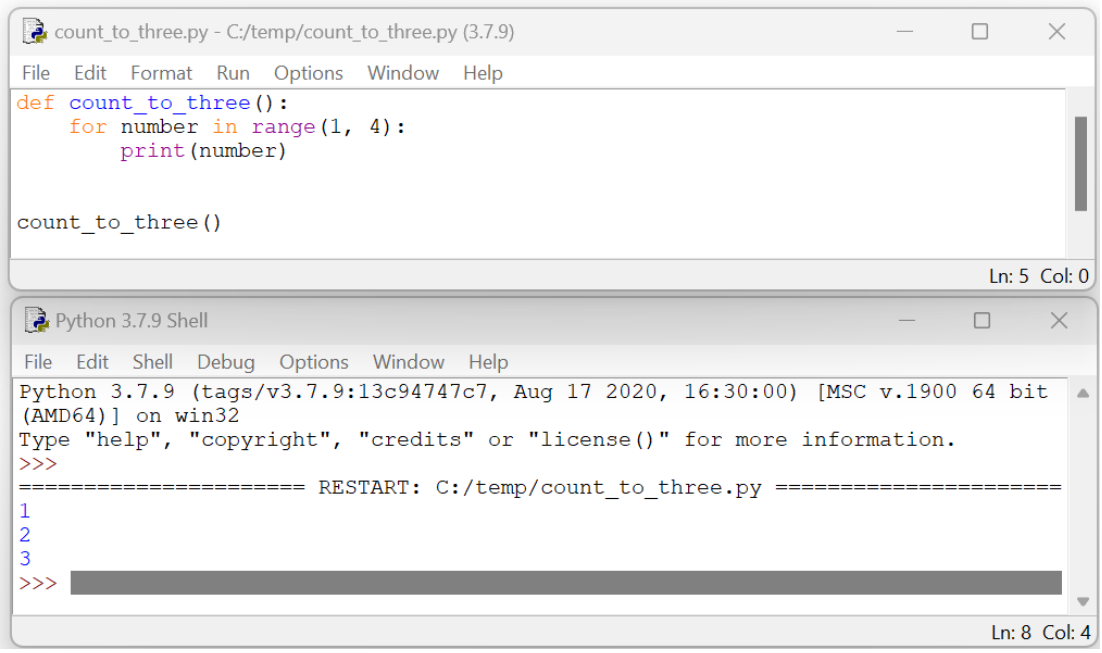

Figure 69. Counting to three in a function

So, that's how your `count_to_three` function works! It's like a mini-adventure where Python gets to count out loud. By writing this function, you've just taught Python a new trick. Great job!

Adding a Twist: Using Parameters in Functions

Great job learning about functions! Now, let's add a little twist by using something called "parameters." Parameters are like special clues you give to a function to tell it what to do differently each time.

What are Parameters?

- Think of a parameter like a secret code you pass to a function to change what it does. It's like when you play a game, and the rules change a little bit each time.

Creating a Function with a Parameter:

1. **Making a Greeting Function:**

   - Let's write a function that says hello in different ways. We'll use a parameter to tell the function who to say hello to.
   - Type this code:

Figure 70

```
def say_hello_to(name):
    print("Hello, " + name + "!")
```

   - Here, `name` is the parameter. It's like a placeholder for the name you want to use when you call the function.

2. **Using Your Function with Different Names:**

   - Now let's use this function. Below your function, type:

Figure 71

```
say_hello_to("Emily")
say_hello_to("Carlos")
```

   - Each time you call `say_hello_to`, you can give it a different name. Python will then use that name in the greeting.

3. **What Happens When You Run It:**

   - When you run this code, Python will say hello to Emily and then to Carlos:

Figure 72

```
Hello, Emily!
Hello, Carlos!
```

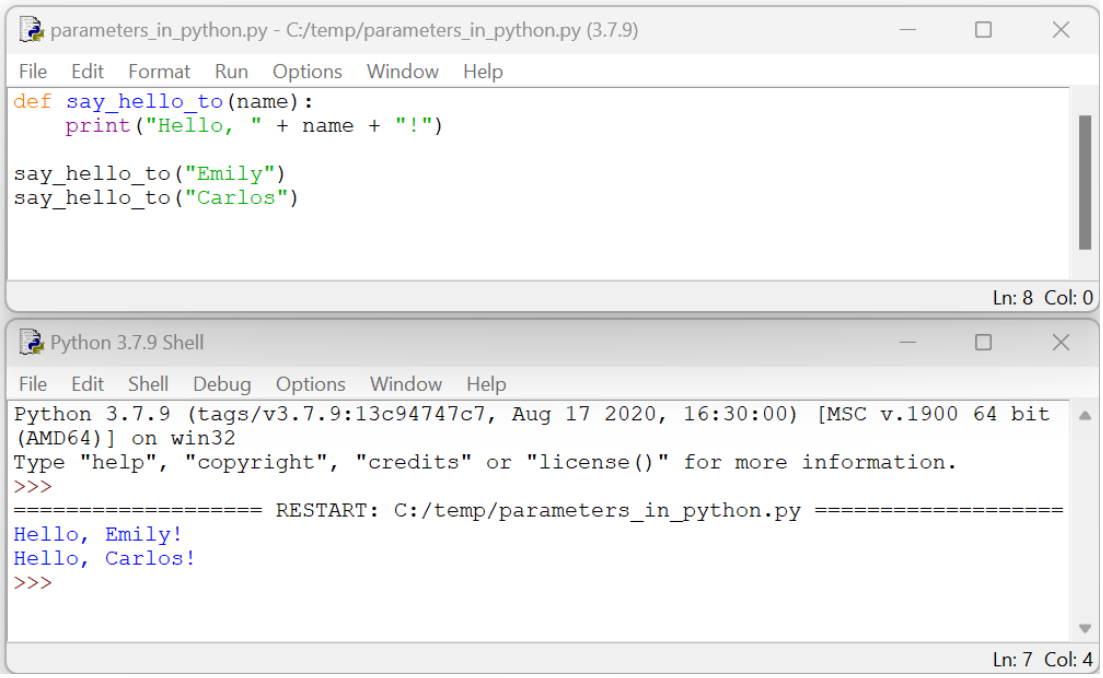

Figure 73. Running greetings function

Why Are Parameters Useful?

- Parameters make functions more flexible and fun. You can use the same function to do slightly different things – like saying hello to different people.

Another Fun Example:

- How about a function to add any two numbers? Let's try it:

Figure 74

```
1 def add_numbers(number1, number2):
2     total = number1 + number2
3     print("The total is:", total)
```

- Here, `number1` and `number2` are parameters. You can use `add_numbers(5, 3)` or `add_numbers(10, 20)` to add different numbers together.

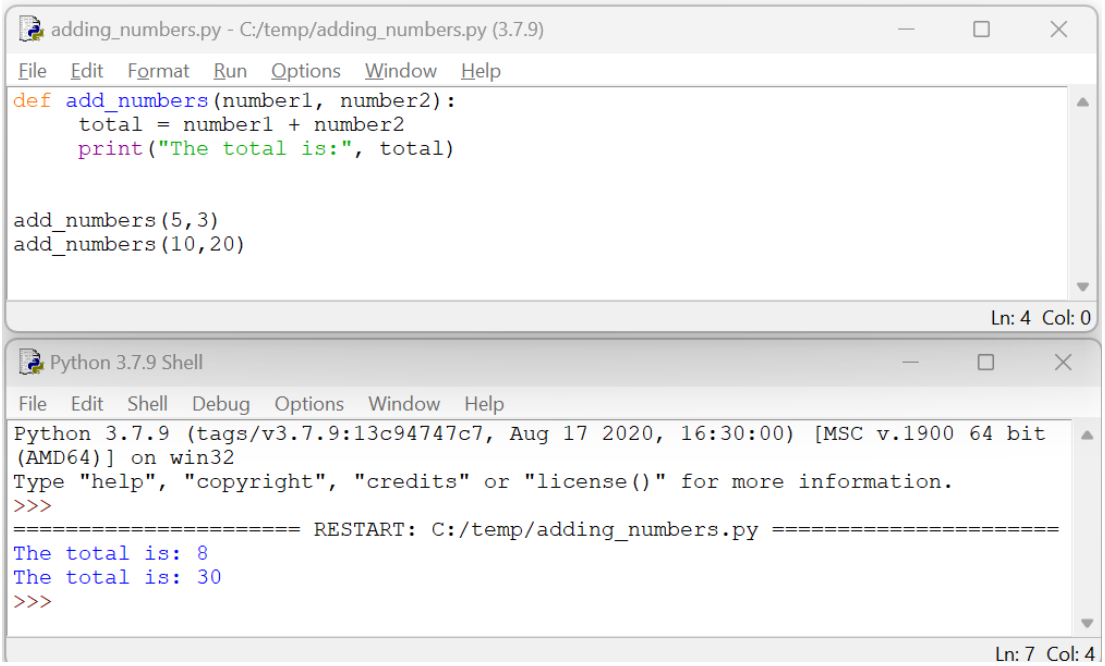

Figure 75. Add Numbers in a function

Using parameters in functions is like giving your functions superpowers. They can do more things and be even more helpful. Try making your own functions with parameters and see what cool things you can make Python do!

You just learned how to write your own functions in Python! Functions are like little helpers that make your coding easier and more fun. Remember, whenever you find yourself doing the same thing many times, a function might be just what you need. Keep experimenting with different functions and see what amazing things you can make Python do!

Chapter 7: Making Games in Python

Figure 76

Building a Simple Game

Guess what? You can make your own games with Python! Let’s dive into the magic of game-making. We’ll start by creating a simple “Guess the

Number” game. It’s fun and easy, and you’ll learn more cool things about Python!

Designing Our Guess the Number Game:

- In this game, Python will think of a number, and you have to guess what it is. Python will tell you if your guess is too high, too low, or just right!

1. Setting Up the Game:

- First, let’s set up our game in a new Python file. We’ll need to tell Python to choose a random number. At the top of your file, type:

Figure 77

```
1 import random
2 secret_number = random.randint(1, 10)
```

- This code lets Python pick a random number between 1 and 10.

2. Guessing the Number:

- Now, let’s ask the player (that’s you!) to guess the number. Add this code:

Figure 78

```
1 print("I am thinking of a number between 1 and 10.")
2 guess = int(input("Can you guess it? "))
```

- This tells Python to ask for a guess and wait for your answer.

3. Checking the Guess:

- Next, we’ll use an if-statement to see if the guess is correct, too high, or too low. Add this:

Figure 79

```
if guess == secret_number:
    print("You got it!")
elif guess < secret_number:
    print("Your guess is too low.")
else:
    print("Your guess is too high.")
```

- These lines check your guess against the secret number and tell you how you did.

4. Running Your Game:

- Save your file as something like `guess_the_number.py`.
- Press `F5` to run your game and try guessing the number!

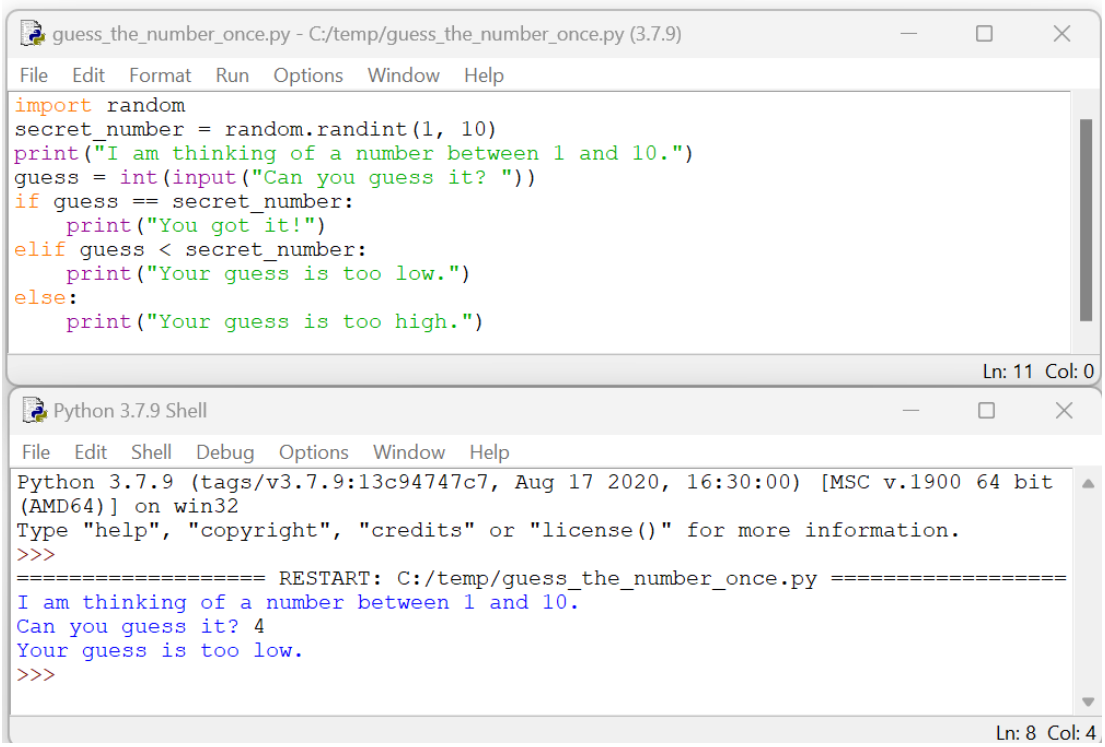

Figure 80. The number guess game

Playing Again and Again:

- Right now, you can only guess once. But you can add a loop to let you keep guessing until you get it right. That's a bit trickier, but super fun to try!

Making Our Game Even More Fun with a Loop!

Let's go ahead and add a loop to our game. Remember, a loop is like a repeating path; it lets you do something over and over again until you want to stop.

Adding a Loop to Keep Guessing:

- We'll use a special kind of loop called a `while` loop. This loop keeps working until a certain condition is met. In our game, the loop will keep going until you guess the number correctly.

Here's How to Add the Loop:

1. **Start with a `while` Loop:**
    - Right before where you ask for a guess, let's add a `while True:` line. It will look like this:

Figure 81

```
while True:
    guess = int(input("Can you guess it? "))
    ...
```

    - `while True:` tells Python to keep repeating everything indented under it until we say stop.
2. **Check the Guess Inside the Loop:**
    - We'll keep the if-statement inside the loop. If your guess is wrong, Python will go back to the start of the loop and ask for another guess.
    - If your guess is right, we need to tell the loop to stop.

3. **Stopping the Loop When You Guess Right:**

    - When you guess the number, we want to congratulate you and then stop the loop. We do this with a command called `break`. Add `break` under the “You got it!” print statement, like this:

Figure 82

```
if guess == secret_number:
    print("You got it!")
    break
```

The Complete Looping Game Code: Here’s how your game code should look now:

Figure 83

```
import random
secret_number = random.randint(1, 10)

print("I am thinking of a number between 1 and 10.")

while True:
    guess = int(input("Can you guess it? "))
    if guess == secret_number:
        print("You got it!")
        break
    elif guess < secret_number:
        print("Your guess is too low.")
    else:
        print("Your guess is too high.")
```

Running Your New Looping Game:

- Save your file and run it again.
- Now you can keep guessing until you find the right number. Each time you guess, Python tells you if you’re too high or too low and then lets you guess again.

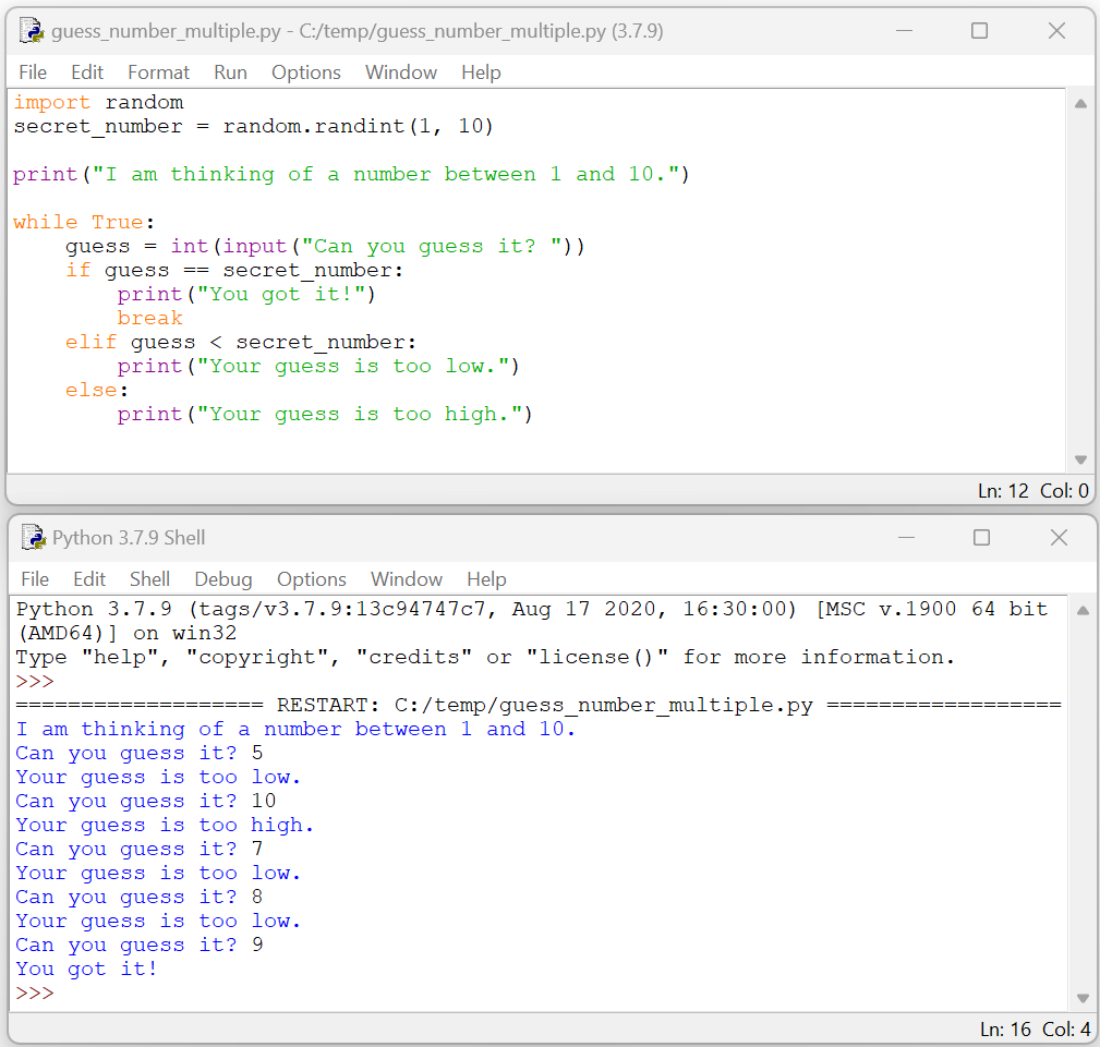

Figure 84. Alt text

With this loop in your game, you can have even more fun trying to guess the secret number. Remember, loops are super useful in coding, especially in games. They let you repeat something without having to write it out every single time. Happy guessing!

And there you go! You just made your very own “Guess the Number” game with Python. Isn’t it amazing how you can turn your ideas into a fun game? Remember, with programming, you can build all kinds of games, so keep experimenting and playing with your code!

Now that we have one game under our belt, let's use some of the same concepts to build another fun game!

Embarking on a Python Adventure in the Forest

Welcome, young explorers! Today, we're going to dive into a new Python game called "The Looping Forest Adventure." In this game, you'll wander through a magical forest in search of hidden treasure. You'll make choices that lead you down different paths, and the adventure will keep going until you find the treasure or decide to take a break. Let's discover how this game works and learn more about Python coding along the way!

Understanding the Game:

When you start the game, you'll be in a dense forest. You can choose to go left or right. If you go left, you might find a cave, and if you go right, you might come across a river. Each choice leads to new decisions, like entering the cave or crossing the river. The game continues until you find the treasure.

Let's create the code one step at a time:

Step-by-Step: Crafting Your Python Treasure Hunt

Hello, young coders! Ready for an exciting adventure? Today, we're going to create a game called "The Looping Forest Adventure" using Python. This game is a treasure hunt in a mysterious forest, where you make choices to find hidden treasure. Let's build this game together, step by step!

1. Setting the Scene:

- First, we need to tell the player what the game is about. We do this with `print` statements. Type these lines in your Python editor:

Figure 85

```
1 print("Welcome to the Looping Treasure Hunt Adventure!")
2 print("You are in a dense forest, searching for the
```

```
3         hidden treasure.")
```

- These lines are like the opening pages of our adventure story. They introduce you to the game.

2. Preparing for the Loop:

- Before we start our adventure, we need to set up a way to keep track of whether we've found the treasure. Type this line:

Figure 86

```
1 treasure_found = False
```

- This is like having a treasure map with a big question mark. It means, "We haven't found the treasure yet."

3. Starting the Adventure Loop:

- Now, let's create a loop that keeps our adventure going. Type this:

Figure 87

```
1 while not treasure_found:
```

- This `while` loop will keep repeating our adventure steps until we find the treasure.

4. Making the First Choice:

- Inside our loop, let's ask the player if they want to go left or right. Type these lines:

Figure 88

```
1 choice = input("Do you want to go left or right?
2                  (left/right) ")
```

- This is where the adventure starts to branch out. Your decision will lead to different paths.

5. Exploring the Left Path:

- Let’s see what happens if the player chooses left. Add these lines:

Figure 89

```
if choice == "left":
    print("You've stumbled upon a hidden cave!")
    choice = input("Do you dare to enter the cave?
                    (yes/no) ")
```

- Here, if you choose left, you find a cave! Then you decide whether to go inside.

6. Finding the Treasure in the Cave:

- If you decide to enter the cave, maybe you’ll find the treasure! Add this:

Figure 90

```
    if choice == "yes":
        print("Inside the cave, you found the treasure!
              Congratulations!")
        treasure_found = True
```

- These lines check if you chose to enter the cave. If you did, you might find the treasure, and the game will end.

7. Exploring the Right Path:

- What if you choose to go right? Let’s write that part. Add these lines:

Figure 91

```
elif choice == "right":
    print("You come across a swiftly flowing river.")
```

- This part starts the adventure on the right path, where you find a river.

8. Deciding at the River:

- At the river, you have to make another decision. Add these lines:

Figure 92

```
    choice = input("Do you wish to swim across or follow
                    the river? (swim/follow) ")
```

- Now you choose: swim across the river or follow it to see where it leads.

9. The Outcomes of the River Path:

- Let's add what happens for each choice at the river. Add this for swimming:

Figure 93

```
    if choice == "swim":
        print("Bravely swimming across, you reach a
                deserted beach, but no treasure here.")
```

- And this for following the river:

Figure 94

```
    else:
        print("Following the river, you find a bridge
                leading to a mysterious temple.")
```

10. The Mysterious Temple:

- If you follow the river and find the temple, you have one more choice. Add this:

Figure 95

```
        choice = input("Do you cross the bridge?
                        (yes/no) ")
        if choice == "yes":
            print("In the temple, you find the hidden
                    treasure! Your quest is complete!")
            treasure_found = True
```

- This is the final decision: if you cross the bridge and enter the temple, you might just find the treasure!

Putting it all together

Great job following along with each step! Now that we've explored the individual parts of our game – setting the scene, creating choices, and imagining the different paths – it's time to put it all together. By combining these elements, we'll see how our choices intertwine and lead us through the exciting twists and turns of our Looping Forest Adventure. Let's bring our adventure to life and see how all these pieces form a complete, fun-filled treasure hunt game in Python. Ready to see your coding work in action? Let's dive in and assemble our game!

Figure 96

```
treasure_found = False
while not treasure_found:
    choice = input("Do you want to go left or right?\
                    (left/right) ")

    if choice == "left":
        print("You've stumbled upon a hidden cave!")
        choice = input("Do you dare to enter the cave? \
                        (yes/no) ")
        if choice == "yes":
            print("Inside the cave, you found the \
                    treasure! Congratulations!")
            treasure_found = True
        else:
            print("You chose to stay outside. The \
                    forest awaits your next move.")
    elif choice == "right":
        print("You come across a swiftly flowing river.")
        choice = input("Do you wish to swim across or \
                        follow the river? (swim/follow) ")
        if choice == "swim":
            print("Bravely swimming across, you reach a \
                    deserted beach, but no treasure \
                    here.")
        else:
            print("Following the river, you find a \
                    bridge leading to a mysterious \
                    temple.")
            choice = input("Do you cross the bridge? \
```

```
                                (yes/no) ")
            if choice == "yes":
                print("In the temple, you find the \
                      hidden treasure! Your quest \
                      is complete!")
                treasure_found = True
            else:
                print("You decide not to cross. The river\
                        path stretches endlessly before \
                        you.")
    else:
        print("Lost in the forest, you decide to stop \
                          and rest. Maybe try another \
                          path next time.")
```

Your Adventure Awaits

Now that we've written our game, it's time to play! Save your Python script and run it. Each decision you make leads you on a different path in the forest. Will you find the treasure, or will you discover other exciting adventures? It's all up to you!

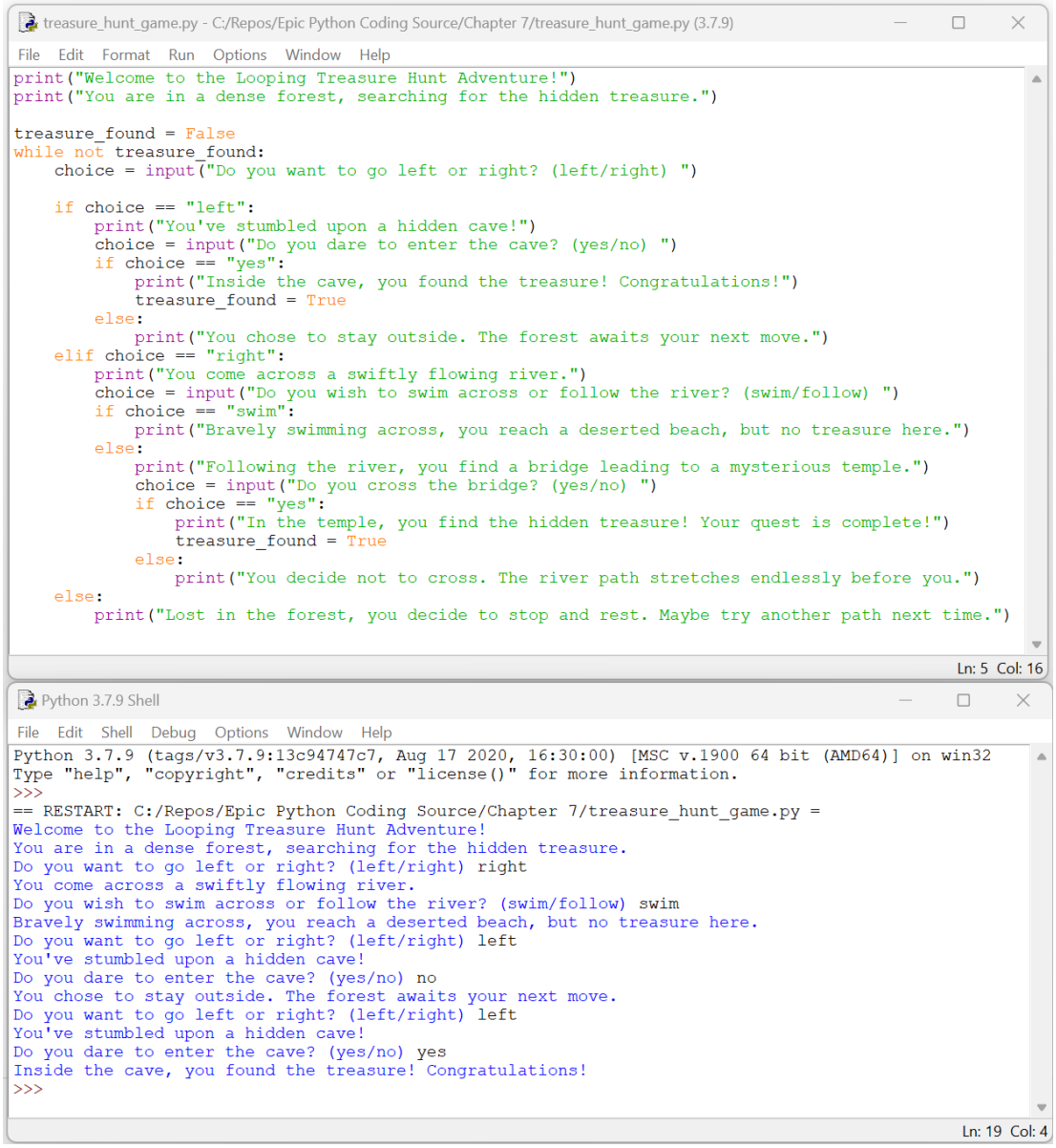

Figure 97. Playing the Treasure Hunt Game

Chapter 8: Python in the World

Figure 98

You've learned so much about Python already! But did you know Python is used by people all over the world to do amazing things? Let's discover some cool ways Python is used in real life and try a simple project to see Python's magic at work!

Python in Action Around Us:

Space Explorations: Reaching for the Stars with Python

- Did you know that space agencies like NASA use Python for their space missions? Python helps them control spacecraft and analyze data from other planets and stars. For example, Python was used in the development of a software system that captured stunning images of a black hole! This is like using Python as a cosmic camera!

Movie Magic: Creating Wonders on Screen

- In the world of movies, Python is a real superstar. It's used to create those breathtaking special effects that make dragons fly and superheroes leap across buildings. An interesting fact: the movie studio behind some of the Harry Potter films used Python to help make those magical scenes come to life. Python helped them animate the enchanting movements of magical creatures!

Gaming Adventures: Building Exciting Worlds

- Many of your favorite video games are powered by Python. Game developers use Python to design levels, create characters, and even make them move and interact. It's like Python is the secret ingredient that brings the gaming world to life. Imagine using Python to create your own game world one day!

Science Discoveries: Unraveling Nature's Mysteries

- Python is a tool for scientists to learn more about our planet and beyond. They use it to predict the weather, study how plants grow, and even explore the mysteries of the ocean. For instance, Python helps scientists understand how climate change affects different ecosystems. It's like Python is a detective solving nature's biggest puzzles!

Isn't it amazing how Python is used in so many exciting ways? From exploring outer space to creating movie magic and video game adventures, Python is everywhere. And scientists rely on it to make important

discoveries about our world. Every time you code in Python, you're learning skills that can help you do incredible things too!

Your Project: Create Your Weather Program:

Now, let's use Python to make your very own weather program!

1. **Getting Ready:**
    - Open a new Python file. This time, you'll ask someone to tell you the weather, and then your program will respond!
2. **Writing the Weather Program:**
    - Start by typing this code:

Figure 99

```
weather = input("What is the weather like today? \
                    (sunny, rainy, snowy) ")
if weather == "sunny":
    print("It's a sunny day! Don't forget your \
            sunglasses.")
elif weather == "rainy":
    print("It's raining. You'll need an umbrella!")
elif weather == "snowy":
    print("Wow, it's snowing! Bundle up in your
warmest \
              clothes.")
else:
    print("Hmm, I don't know that weather. Sounds like
\
              an adventure!")
```

    - This program asks for the weather and gives a response based on what you type.
3. **Trying Out Your Weather Station:**
    - Run your program and type in different weather conditions like "sunny," "rainy," or "snowy" and see what your program says.

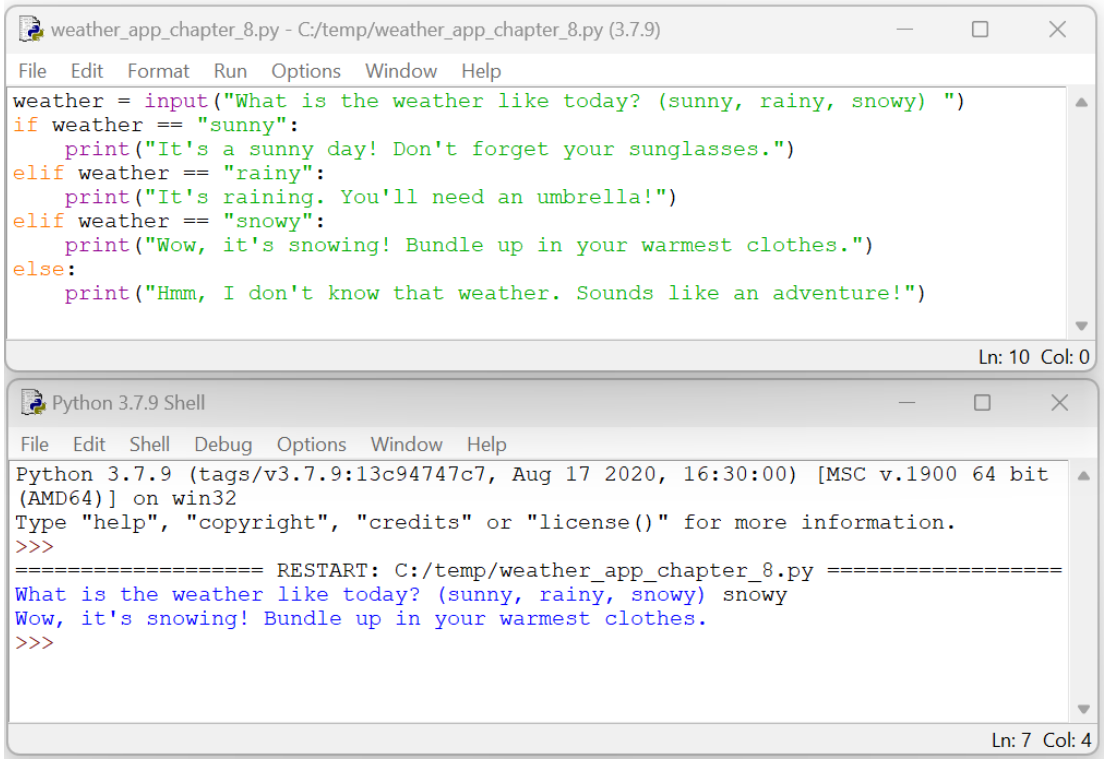

Figure 100. The weather app in action

Roll Call with Python - Building an Attendance Tracker Step by Step

Introduction to Our Python Project

Hey there, young coders! Have you ever thought about how teachers keep track of who's in class? Today, we're going to create our own Attendance Tracker using Python! We'll build it step by step, learning about lists, loops, and counting. In the end, we'll put all our pieces together to make a cool program. Let's get started!

1. Creating a List of Students:

First, we need a list of students for our roll call. In Python, we make a list using square brackets `[]`, and we put names inside it, separated by

commas.

Imagine we have five friends: Alice, Bob, Charlie, David, and Eva. Our list will look like this:

Figure 101

```
students = ["Alice", "Bob", "Charlie", "David", "Eva"]
```

2. Setting Up Counters for Attendance:

Next, we need two counters: one for students who are present and one for those who are absent. We'll start them both at 0 because we haven't counted anyone yet.

Figure 102

```
present_count = 0
absent_count = 0
```

3. Asking About Each Student:

Now we're going to ask if each student is here. We use a loop for this. As we talked about, a loop is like going around in a circle, doing the same thing for each item in our list.

We're going to use a `for` loop, which lets us do something for each student in our `students` list.

Figure 103

```
students = ["Alice", "Bob", "Charlie", "David", "Eva"]

# Variables to keep track of attendance
present_count = 0
absent_count = 0

# Loop through each student in the list
for student in students:
    # ask each student if they are present
```

4. Checking if a Student is Present:

Inside our loop, we'll ask if each student is present. We wait for an answer, and if the answer is 'yes', we add 1 to our `present_count` variable. If the answer is 'no', we add 1 to our `absent_count` variable.

We use an `if` statement to choose whether or not to mark a student absent or present in the total. An `if` statement lets us choose what to do based on the answer.

Figure 104

```
    # determine if a student is present
    response = input(f"Is {student} present? (yes/no): ")
                     .lower()

    # Check if the student is present
    if response == "yes":
        print(f"{student} is present.")
        present_count += 1
    else:
        print(f"{student} is absent.")
        absent_count += 1
```

We want an easy way to ask the user if each student is present. That is where we use a fancy Python trick called *String Interpolation*.

Understanding String Interpolation with a Fun Example

We need a way for our Attendance Tracker asks questions like, "Is Alice present? (yes/no): "? It needs to know how to change the name for each student, right? That's where string interpolation comes in. It sounds like a big term, but it's actually pretty simple and cool!

Imagine you have a sentence written on a piece of paper: "Is _____ present?" Now, imagine you have stickers with names on them. You can stick a name into that blank spot to ask about different students. In Python, we do something similar with our code!

What is String Interpolation?

- **String**: This is just a fancy word for text in Python.
- **Interpolation**: This means putting something in the middle of something else.

So, string interpolation is like putting a piece of information (like a name) into the middle of a string (a sentence or text) in Python.

How We Use It in Our Program:

Here is the code for string interpolation used on each student name `f"Is {student} present? (yes/no): "`, we are telling Python to make a special kind of string (that's what the `f` before the quotes is for). The `{student}` part is like a blank space where Python can put each student's name.

For each student in our list, Python replaces `{student}` with the actual name, just like we might use stickers to fill in the blank space on our paper.

Example:

- When Python sees `f"Is {student} present? (yes/no): "` and `student` is "Alice", it turns into: "Is Alice present? (yes/no): "
- Then when the `student` is "Bob", it turns into: "Is Bob present? (yes/no): "
- And so on for each student!

Summing Things Up

Once we've finished taking attendance, we get to share what we found. We'll use what we learned earlier to write a message that tells everyone how many students are here and how many are not. It's like putting the final pieces of our puzzle together to see the whole picture! Did you catch that we are using string interpolation again to help display the results?

Figure 105

```
# Print the total number of students present and absent
# Again we are using string interpolation
print(f"\nTotal Present: {present_count}")
print(f"Total Absent: {absent_count}")
```

5. Putting It All Together:

Let’s put all these pieces together to make our complete Attendance Tracker. Here’s what our whole program will look like:

Figure 106

```
# List of students in the class
students = ["Alice", "Bob", "Charlie", "David", "Eva"]

# Variables to keep track of attendance
present_count = 0
absent_count = 0

# Loop through each student in the list
for student in students:
    # use string interpolation to prompt
    # a particular student
    response = input(f"Is {student} present? (yes/no): ")
                    .lower()

    # Check if the student is present
    if response == "yes":
        print(f"{student} is present.")
        present_count += 1
    else:
        print(f"{student} is absent.")
        absent_count += 1

# Print the total number of students present and absent
# Again we are using string interpolation
print(f"\nTotal Present: {present_count}")
print(f"Total Absent: {absent_count}")
```

Understanding How We Count Students in Our Program

In our Attendance Tracker, we keep track of how many students are present and how many are absent. To do this, we use a special trick in Python. You

might have seen `present_count += 1` in our code. Let's break that down to understand what it does.

What Does `present_count += 1` Mean?

- Think of `present_count` as a jar where we keep marbles. Each marble represents one student who is present.
- The `+= 1` part is like adding one more marble to the jar.
- So, every time we say `present_count += 1,` we're adding a marble to the "present" jar because another student is here.

Example:

- Imagine the jar is empty in the beginning (`present_count = 0).`
- When Alice is present, we add a marble. Now, there's 1 marble (`present_count = 1).`
- If Bob is also present, we add another marble. Now, there are 2 marbles in the jar (`present_count = 2).`
- This keeps going for each student who is present.

What About `absent_count += 1`?

- We do the same thing for `absent_count.` This is another jar, but for students who are absent.
- Each time a student is absent, we add a marble to this "absent" jar.

So, `present_count += 1` and `absent_count += 1` are our ways of keeping count of how many students are present and how many are absent. It's like adding marbles to two different jars to keep track of who's here and who's not. This makes our Attendance Tracker smart and helpful!

---

6. Running Our Program:

After typing in the final code, save it, and run it using F5. Your computer will start asking you about each student. Type 'yes' if they are present, and

‘no’ if they are not. In the end, you’ll see how many are present and how many are absent.

take_attendance.py - C:\Repos\Python For Beginners\Source\Chapter 8\take_attendance.py (...

File Edit Format Run Options Window Help

```
# List of students in the class
students = ["Alice", "Bob", "Charlie", "David", "Eva"]

# Variables to keep track of attendance
present_count = 0
absent_count = 0

# Loop through each student in the list
for student in students:
    response = input(f"Is {student} present? (yes/no): ") \
        .lower()

    # Check if the student is present
    if response == "yes":
        print(f"{student} is present.")
        present_count += 1
    else:
        print(f"{student} is absent.")
        absent_count += 1

# Print the total number of students present and absent
print(f"\nTotal Present: {present_count}")
print(f"Total Absent: {absent_count}")
```

Ln: 1 Col: 0

Python 3.7.9 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 16:30:00) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Repos\Python For Beginners\Source\Chapter 8\take_attendance.py ==
Is Alice present? (yes/no): yes
Alice is present.
Is Bob present? (yes/no): no
Bob is absent.
Is Charlie present? (yes/no): no
Charlie is absent.
Is David present? (yes/no): yes
David is present.
Is Eva present? (yes/no): yes
Eva is present.

Total Present: 3
Total Absent: 2
>>>
```

Ln: 18 Col: 4

Figure 107. Taking Attendance

Great job! You just made a program that can help teachers with roll call.

Python can be used for so many cool things, like finding out about the weather or taking attendance in class! By making this weather program and attendance taker, you're learning to use Python just like real-world scientists and programmers. Keep exploring and having fun with Python – there's so much more to discover!

Conclusion: Your Python Journey Continues

Figure 108. Graduation)

Congratulations on Your Amazing Python Adventure!

Wow, you've learned so much about Python! From drawing shapes with Turtle to making your own games, you're now a real Python explorer. But guess what? Your adventure doesn't stop here. There's still so much more to learn and explore with Python. Let's talk about your next steps and how you can continue your exciting journey.

Next Steps in Learning Python:

1. **Keep Practicing:**
    - The best way to get better at Python is to keep practicing. Try changing the games you made, or maybe even create new ones. Remember, practice makes perfect!
2. **Find More Python Projects:**
    - There are lots of fun Python projects online that are perfect for beginners. You can make things like calculators, simple chatbots, or even a small website.
3. **Join a Coding Club:**
    - Many schools and communities have coding clubs where you can learn with other kids. It's a fun way to learn new things and make friends who also love coding.
4. **Check Out Books and Websites:**
    - There are many cool books and websites that can help you learn more about Python. Some websites even have interactive Python exercises and puzzles!

Resources for Further Learning:

- **Websites:**
    - Code.org and Scratch are great websites to practice coding with fun projects.

    - Python.org has lots of resources and is where you can download the latest version of Python.
- **Books:**
    - Look for books that are made for kids learning Python. They usually have fun examples and easy-to-understand instructions. When you become more advanced and want to use python to create real video games, check out Creating Video Games using PyGame. PyGame is a library that makes it easy to create games in python!

Encouragement to Keep Exploring:

- Remember, learning to code is like going on a treasure hunt. Sometimes it's challenging, but it's always exciting. And the more you learn, the more treasures you'll find. Python is a language that can help you create amazing things, solve problems, and have a lot of fun.
- Don't be afraid to try new things and make mistakes. That's how you learn and grow. You're already on your way to becoming a great programmer!

Your Python journey is just beginning, and there are endless possibilities waiting for you. Keep exploring, keep learning, and most importantly, keep having fun with Python. You're going to do amazing things!

Appendices

Accessing the Source Code for “Epic Python Coding”

Find All the Code You Need Online!

Did you have fun learning Python with “Epic Python Coding”? Want to dive deeper and play around with the code from the book? Great news! All the source code for the projects and examples in this book is available online. You can find, download, and even modify the code to try your own ideas!

Where to Find the Code:

- **GitHub Repository:** All the source code is hosted on GitHub, a website where programmers share their code. You can visit our special repository (a place where we keep all our code) to find everything you need.
- **URL:** To access the code, simply go to Epic Python Coding Source on GitHub.

How to Use the Repository:

1. **Visit the Link:** Click on the link or type it into your web browser.
2. **Explore the Files:** You’ll see a list of files, each named after a chapter or a specific project from the book.
3. **View or Download Code:** Click on a file to view the code. You can also download the files to your computer.

Experiment with the Code:

- Feel free to play with the code! You can try changing it and see what happens. This is a great way to learn more and have fun with Python.

- If you’re not sure how to do something, ask a parent, teacher, or friend for help.

Keep Coding and Exploring:

- Remember, the more you practice, the better you’ll get. Use the code from the book as a starting point for your own Python adventures.

We hope that having access to the source code helps you on your journey in learning Python. Keep exploring, keep experimenting, and most importantly, keep enjoying the magic of coding!

---

1. Glossary of Terms

1. **Variable:** A storage space in programming where data (like numbers or text) is kept.

2. **String**: A string is like a word or a sentence. It’s just a bunch of letters, numbers, or other symbols stuck together. In Python, you show you’re using a string by putting it in quotes.

3. **Number**: Numbers in Python are just like numbers you use in math class. They can be whole numbers (like 1, 2, 3) or numbers with decimals (like 1.5 or 2.75).

4. **Function:** A block of code that performs a specific task and can be reused.

5. **Loop:** A sequence of instructions that repeats until a certain condition is met.

6. **Conditional Statement:** A command that performs different actions depending on whether a condition is true or false.

7. **Algorithm:** Like a recipe for solving a problem. It gives step-by-step instructions on what to do.

8. **List:** A line of boxes in your computer, where each box holds something like a number or a word.

9. **Bug:** A mistake in a computer program that makes it act weird or wrong.

10. **Class:** A blueprint for making objects in computer programming. It tells the computer what information and actions an object should have.

11. **Comment:** In a computer program, comments are like little notes that help explain what the code is doing. They are just for people to read and don't affect how the program runs. In Python, if you want to write a comment, you start it with the # symbol. Anything after the # on that line is a comment.

12. **Data Type:** A kind of information that a program can use or change. It can be numbers, decimal numbers, or words.

13. **Exception:** A kind of mistake that happens while the computer is running a program, like trying to divide a number by zero.

14. **IDE (Integrated Development Environment):** This is like a big box of tools that helps you write and fix your computer code. Think of it as a special kind of computer program where you can write code, test it, and make sure it works right. A good example of an IDE is IDLE, which you get when you install Python. It's like having a helpful assistant for coding!

15. **Library:** A collection of ready-made code that makes it easier to do common things in programming without starting from scratch. For instance, in Python, the `random` library is used to generate random numbers.

16. **Object:** Something created in programming that can store information and do actions. It’s made using a class as a blueprint.

17. **Operator:** A symbol that tells the computer to do math or make decisions. Like + for adding.

18. **Syntax:** The rules for writing code that the computer understands. It’s like grammar for programming.

19. **Variable Scope:** Where in a program a piece of information (a variable) can be used. Some information can be used everywhere (global) and some only in specific places (local).

20. **While Loop:** A part of a program that keeps doing the same thing over and over as long as a certain condition is true.

---

2. Extra Python Challenges

Below are several engaging project ideas to help enhance your Python programming skills:

- **Basic Calculator:** Create a simple calculator that can add, subtract, multiply, and divide.

- **Story Generator:** Write a program that randomly generates funny stories by mixing up user input.

- **Weather Forecaster:** Create a program that asks for the user’s city and then uses an API to display the current weather in that city.

- **Mad Libs Game:** Similar to the Story Generator, but this time, the program asks the user for specific types of words (like a noun, verb, adjective) and fills them into a pre-written story template.

- **Rock, Paper, Scissors Game:** A program where the user can play the classic rock, paper, scissors game against the computer.

- **Simple Quiz Game:** Write a program that asks the user a set of multiple-choice questions and then scores them at the end.
- **Turtle Race:** Using the Turtle graphics library, create a simple animation where multiple turtle objects race across the screen.
- **Times Table Tester:** A program that quizzes the user on multiplication tables.
- **Word Counter:** A tool that takes a block of text (like a paragraph) and counts how many times each word appears.
- **Alarm Clock:** A simple alarm clock where the user can set a time, and the program alerts them when it's reached.
- **Currency Converter:** A program that converts amounts from one currency to another using real-time exchange rates from an API.

Resources

Websites:

1. **CodeCombat:** This is an interactive gaming website where kids learn Python by writing code to navigate through different levels in a game.
2. **Trinket.io:** A simple online Python editor perfect for beginners. It allows kids to write Python code and see the results immediately.
3. **Python for Beginners:** Although not exclusively for kids, this website has very straightforward Python tutorials that are easy to understand.

Parent/Teacher Guide

Python is an excellent programming language for children to learn due to its readability and versatility. As a parent or teacher, your support is crucial in nurturing a child's interest and skills in programming. This guide offers

strategies and tips to facilitate children's learning of Python programming effectively.

Understanding Python

Before assisting children, it's helpful to have a basic understanding of Python yourself. Python is known for its simplicity and readability, making it a great first language for children. Familiarize yourself with basic concepts like loops, variables, and functions.

Tips and Strategies

1. Start with the Basics

- **Simplify Concepts:** Break down programming concepts into simple terms. Compare loops to everyday tasks, like brushing teeth daily, or variables to boxes that hold different things.
- **Use Visual Aids:** Diagrams, flowcharts, and visual programming tools can make abstract concepts more concrete.

2. Encourage Exploration

- **Project-Based Learning:** Encourage building small projects, like a simple calculator or a basic game. This makes learning more engaging and practical.
- **Encourage Questions:** Prompt them to ask 'why' and 'how' questions, fostering a deeper understanding and curiosity.

3. Foster Problem-Solving Skills

- **Debugging Exercises:** Teach them to identify and fix errors in simple pieces of code. This develops critical thinking and resilience.
- **Step-by-Step Approach:** Encourage them to tackle problems one step at a time, rather than getting overwhelmed by the bigger picture.

4. Keep the Learning Fun and Interactive

- **Gamify Learning:** Use programming games and challenges. Platforms like CodeCombat or Scratch offer interactive ways to learn Python.
- **Relatable Examples:** Use examples and projects related to their interests, whether it's sports, art, or video games.

5. Encourage Regular Practice

- **Consistency Over Intensity:** Short, regular coding sessions are more effective than infrequent, long ones.
- **Real-World Applications:** Show them how Python is used in the real world, like in developing video games or in scientific research.

6. Provide Support and Encouragement

- **Celebrate Achievements:** Acknowledge their progress, no matter how small.
- **Be Patient:** Understand that everyone learns at their own pace. Offer encouragement and support rather than pressure.

7. Use Resources Wisely

- **Online Courses and Tutorials:** Utilize free online resources and tutorials that are designed for children.
- **Books and Educational Kits:** Consider age-appropriate books and Python learning kits.

8. Create a Collaborative Environment

- **Group Learning:** If possible, arrange for them to learn with peers. This encourages collaboration and idea-sharing.

- **Parent-Child Coding Sessions:** Engage in coding activities together. This not only helps in learning but also strengthens your bond.

Your involvement in a child's journey in learning Python can be incredibly rewarding. By using these strategies, you can help them develop not only programming skills but also critical thinking, problem-solving, and a lifelong love for learning. Remember, the goal is to foster curiosity and enjoyment in the world of programming.


Epic Python Coding: Interactive Coding Adventures for Kids