Python Coding Level 1

5 Day Camp Plan

Day 1 Rock, Paper, Scissors

Day 2 Word Guessing Game

Day 3 Finish Guessing Game
Storyboarding CYOA

Day 4 Choose your own adventure
ASCII art

Day 5 Wrap-up CYOA Game

4 Day Camp Plan

Day 1 Rock, Paper, Scissors

Day 2 Word Guessing Game

Day 3 Storyboard CYOA

Day 4 CYOA Game

Additional Materials

Printed Storyboards

Typical Day

9:00 - 10:30 (9:00 - 10:15)	**10:30 - 11:30** (10:15 - 11:15)	**11:30 - 12:30** (11:15 - 12:00)	**12:30 - 1:15** (12:00 - 12:45)	**1:15 - 2:15** (12:45 - 1:30)	**2:15 - 3:00** (1:30 - 2:15)	**3:00 - 4:00** (2:15 - 3:00)
Block 1	Morning Break	Block 2	Lunch	Block 3	Afternoon Break	Block 4

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Camp Rules

1. RESPECT. Respect for peers, facility, equipment, and instructors. (No exceptions!)
2. No Foul Language
3. No excessively violent or inappropriate content (“Looney Tunes” rule of thumb)
4. No substituting outdoor breaks for more computer time. If it is raining, play indoor, off-the-devices games.
5. HAVE FUN!!!

Reminders

- Have parents SIGN IN/OUT everyday (unless written permissions say otherwise)
- Leave the computers on at the end of the day to show parents what the kids have done.
- Make sure to have all release forms filled out by parents before the end of the day.
- Do your best to engage the kids wherever possible. Getting up and pointing at the projected screen, asking THEM questions. It is very easy to turn these lessons into lectures but you will lose students that way. Keep it energetic and fun!

We encourage all instructors to teach however necessary to ensure the students understand the material enough to confidently construct projects on their own, however we recommend you stick to the pacing and curriculum below.

We acknowledge there are multiple different ways to get things done however for simplicity-sake, we like to keep the process linear and straight to the point as camp can be quite overwhelming as is.

Should you have a group of advanced students, you are welcome to teach the more complex methods if the students are interested.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 1

Class Notes

- Read the HEALTH INFORMATION FOR EACH CHILD!!! (After name game)
- Spend enough time at the beginning to get over 1st day shyness and learn each other's names
- Be thorough in helping the kids set up their own projects folder
- Many of the kids in this camp probably don't have a lot of experience working with a computer aside from playing games and watching videos. Have patience with them as they learn to develop these basic skills, and let them know it is ok that things don't work out right away, just take a breather and then try again.

Day 1 Schedule

9:00 - 9:30	Camp Welcome and Name Game
9:30 - 10:30 (9:30 - 10:15)	Rock Paper Scissors: Hello World!
10:30 - 11:30 (10:15 - 11:15)	**Morning Break**
11:30 - 12:30 (11:15 - 12:00)	Rock Paper Scissors: User Choice + Decide Winner
12:30 - 1:15 (12:00 - 12:45)	**Lunch Break**
1:15 - 2:15 (12:45 - 1:30)	Rock Paper Scissors: Decide Winner cont. + Game Loop
2:15 - 3:00 (1:30 - 2:15)	**Afternoon Break**
3:00 - 4:00 (2:15 - 3:00)	Rock Paper Scissors: Points
4:00 (3:00)	**Wrap up and dismiss**

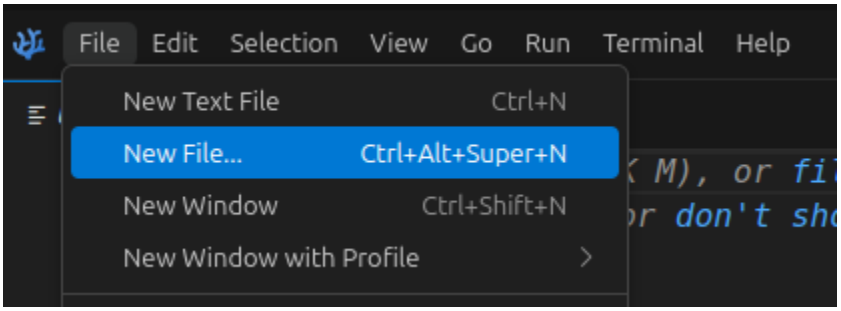
© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 1 - Block 0 - Camp Welcome

- Go over the 'Monday Minders' in the 'Weekly Reminders' guide.
- Have the kids create their own folders in "My Documents" or "Documents" drive before starting anything
- All instructors must go over every camper's release forms some time after the name game but before first break.

Day 1 - Block 1 - Hello Computer, let's play!

Start by opening VSCodium and creating a new Python file in the folder where you are working, and name it rps.py. This file will contain all the code for the Rock-Paper-Scissors game.



First, explain what a variable is. In Python, a variable is like a little box that holds information. You can use variables to store numbers, words, or other data, and you can change what's inside the variable later. For example, let's make a new variable that holds a message. This kind of information is called a *string*, which means a bunch of letters, numbers, or symbols all together inside quotation marks:

intro_msg = "Hello, would you like to play a game?"

Next, we want the computer to show this message in the terminal. To do that, we use Python's built-in **print** function, like this:

print(intro_msg)

To run this code, we open the terminal in our code editor by pressing **Ctrl +** ` (that little backtick key above Tab). Then we type this command to run the Python file:

python rps.py

When you run the file, you should see your message printed in the terminal.

Now let's make the game ask for the player's name. To do that, we can use Python's built-in **input** function. This function allows the player to type something, and Python will save it in a variable. The **input()** function also lets us display a helpful message (called a **prompt**) that tells the player what to do. Here's how we do it:

name = input("What is your name?\n")

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Notice the \n part at the end of the sentence. This means "new line." It tells the computer to move the cursor to the next line after showing the message, so that the player's typing doesn't happen on the same line as the question, it looks neater this way!

After the player types their name, we can make the computer print a message that includes their name. We do this like this:

print("Let's play rock, paper, scissors " + name + "!")
In this line, the + sign doesn't mean "add numbers" like in math. Instead, when used with strings (words), the + sign sticks the pieces of text together. So the computer will take the first string, add the player's name, and then add the "!" at the end, making one long sentence.

Finally, save your file and run it again using:

python rps.py

Now your game will greet the player by name!

Now let's set up the game to allow you to play one round of rock paper scissors.

In Python, there is something called a *list*. A list is used to store several pieces of information all in one place. Since our game has three possible choices—rock, paper, and scissors—we can put these into a list. This way, we don't have to type out all the choices every time we need them. Here's how we can make a list for our game:

moves = ["rock", "paper", "scissors"]

Now the list called **moves** holds the three choices the player and the computer can pick from. This makes it easy for the computer or player to select one of these options later in the game.

Next, let's make the computer pick one of these moves. Python has a built-in tool called the random module that can help us choose something at random. First, we need to import the random module at the top of the file like this:

import random

After that, we can tell the computer to pick a random choice from the moves list like this:

computer_choice = random.choice(moves)

This line tells Python to pick one of the items from the moves list and store it in a variable called **computer_choice.**
Finally, let's print the computer's choice to see what it picked:

print('Computer chooses: ' + computer_choice)

Now you can run your Python file again and see what move the computer picks!

Day 1 - Morning Break

Weather permitting, take them outside and run around (tag, dodgeball, camouflage, etc)

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

If the weather is poor, have them play indoor games that are NOT ON THE LAPTOPS OR THEIR PHONES.

> REMEMBER: ALL STAFF should be outside with the children for EVERY break unless under certain circumstances which have been CLEARED by a manager

Day 1 - Block 2 - Personal Choice + Decide Winner

Now that we've set up the game to allow the computer to make a decision, we also need a way for the player to make their own Rock-Paper-Scissors choice.

To do this, we'll create a new variable to store the player's choice. We can use Python's built-in **input() function** again. This allows the player to type their selection into the terminal.

Let's place this line of code before the computer makes its choice. That way, the player won't know what the computer picked before making their own decision:

```
moves = ["rock", "paper", "scissors"]

player_choice = input("Choose rock, paper or scissors: ")

computer_choice = random.choice(moves)

print('Computer chooses: ' + computer_choice)
```

Now we need to be able to compare both my choice and the computer's choice. We need to first explain python indentation Basics

Python Indentation Basics

Start by explaining how indentation works in Python. Use your own example! Or use the one given below:

1. Clean your Room
    a. Make your bed
    b. Put clothes in your closet
2. Finish Homework
    a. Do your fractions worksheet
    b. Read 1 Chapter of your book

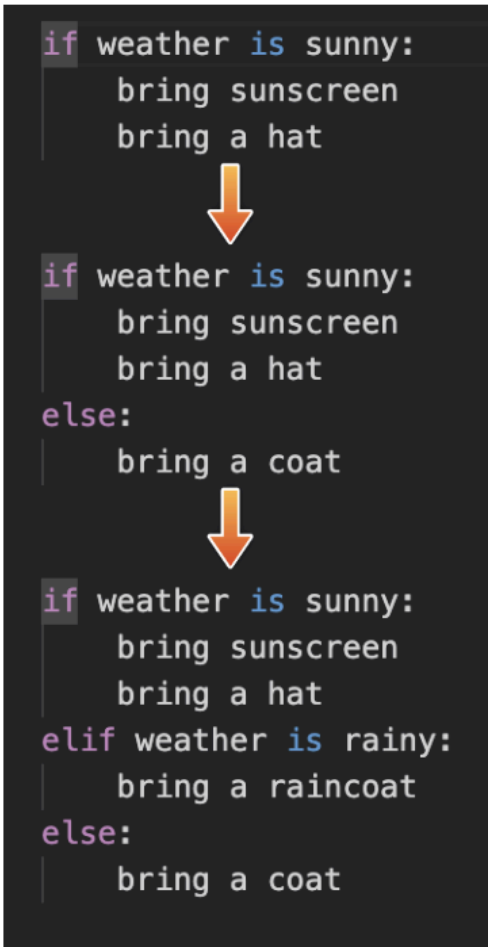
Explain that if else statements work like a todo list. If you clean your room, what are you going to do? How do you know that? So that means that anything below an if statement and moved to the left is a result of the statement.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

If else in Python

Now that they have indentation down, explain how if else works in python.

Start by going over how an if else statement works, then build it into an if, else if, else statement. This should ideally be done on a whiteboard.. Feel free to create your own example! Otherwise feel free to use one based on the weather. Make sure to reiterate that each action is a result of the above statement. Make sure to explain what an elif is!



Back to the Rock Paper Scissors Game

The first situation we need to check is whether the player and the computer made the same choice, which would mean the round is a tie. We can do this with an if statement like this:

```
if player_choice == computer_choice:
    print("It's a tie!")
```

This checks if the player's choice and the computer's choice are the same. If they are, the computer prints "It's a tie!" and returns the word "tie" to let the rest of the program know the result.

After checking for a tie, we want to show the class how to compare different choices made by the player and the computer to decide who wins the round. This is where we use the **elif** statement and the **and** operator.

For example, if the player chooses "rock" and the computer chooses "scissors", then the player wins because rock beats scissors. We can check this with an **elif statement** like this:

```
if player_choice == computer_choice:
    print("It's a tie!")
elif player_choice == "rock" and computer_choice == "scissors":
    print("Rock crushes scissors! You win!")
```

Here, we are asking Python to check two things at the same time: **is the player's choice "rock"? And is the computer's choice "scissors"?** Both of these must be true for this block of code to run. If they are, the computer will print a message saying the player wins, and the function will return the string "player" to show that the player won this round.

Now have them create similar statements using elif for all the other possible combinations.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

After that, let's introduce an **else** statement as part of our **if/elif block**. This will handle any situation where the user enters a choice that is not one of the valid options. If the player types something incorrectly, like a misspelled word, we want the program to print a message letting them know it's not a valid choice.

Add the **else** block at the end of your existing **if/elif statements** like this:

```
else:
    print("Hmm that isn't one of our choices!")
```

Now, if the player accidentally spells one of the choices wrong, the program will display this helpful message!

Day 1 - Lunch Break

Day 1 - Block 3 - Game Loop

Continue working on the if blocks 'if' they aren't finished yet. Try your best to not give them the answers but help them figure them out on their own.

Now that the game can decide who wins a round, let's make it run over and over again until the player wants to stop. This is where we introduce something new: a loop. In Python, a loop lets us repeat a block of code multiple times. The type of loop we'll use is called a **while True loop**. This kind of loop keeps going forever, unless we tell it to stop with a special command (which we'll add later).

We will add the loop below our moves list. Once you've added the loop, you'll need to make sure that all the code that should repeat is part of the loop. To do that, you'll have to move everything below the loop one level over by adding an extra tab (or indentation). This tells Python which lines should be repeated as part of the game loop. Now, every time the loop runs, the player can make a choice, the computer can respond, and the result can be shown again and again.

```
while True:
    player_choice = input("Choose rock, paper or scissors: ")

    computer_choice = random.choice(moves)

    print('Computer chooses: ' + computer_choice)

    if player_choice == computer_choice:
```

Now that the game is running over and over again, let's add a way to see that we've started a new round. Inside and at the top of the loop, we will print a title so the player knows a new round is starting:

print("\n--- Rock, Paper, Scissors ---")

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 1 - Afternoon Break

Day 1 - Block 4 - Points

We will start by introducing a new variable to keep track of the winner each round. At the beginning of every round, set this variable to the string "**none**", which will represent a tie. Later in the round, we'll update this variable based on the outcome.

```
print("\n--- Rock, Paper, Scissors ---")

winner = "none"

player_choice = input("Choose rock, paper or scissors: ")
```

For any situation where there is a clear winner, we simply change the value of the **winner** variable to either **"player"** or **"computer"**. This way, we can check the value later to decide who should receive a point.

Reminder: If at any point you need to stop a program from running in Python use CTRL + C

```
if player_choice == computer_choice:
    print("It's a tie!")
elif player_choice == "rock" and computer_choice == "scissors":
    print("Rock crushes scissors! You win!")
    winner = "player"
elif player_choice == "rock" and computer_choice == "paper":
    print("Paper covers rock! Computer wins!")
    winner = "computer"
elif player_choice == "paper" and computer_choice == "rock":
    print("Paper covers rock! You win!")
    winner = "player"
elif player_choice == "paper" and computer_choice == "scissors":
    print("Scissors cut paper! Computer wins!")
    winner = "computer"
elif player_choice == "scissors" and computer_choice == "paper":
    print("Scissors cut paper! You win!")
    winner = "player"
elif player_choice == "scissors" and computer_choice == "rock":
    print("Rock crushes scissors! Computer wins!")
    winner = "computer"
```

To keep track of the score throughout the game, we'll create two new variables. These should be placed above and outside the while loop, so that they don't reset every round. Name the variables **player_score** and **computer_score**, and set both of them to 0 to start:

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

```
player_score = 0
computer_score = 0
```

Now we introduce something new: the **+= operator**. This is a shortcut in Python for adding to a variable. For example, instead of writing **player_score = player_score + 1**, we can write **player_score += 1**. This means "increase player_score by 1".

We use an **if statement** to check who the winner is. If the player wins, we add 1 to their score. If the computer wins, we add 1 to the computer's score. If it's a tie, nothing changes:

```
if winner == "player":
    player_score += 1
elif winner == "computer":
    computer_score += 1
```

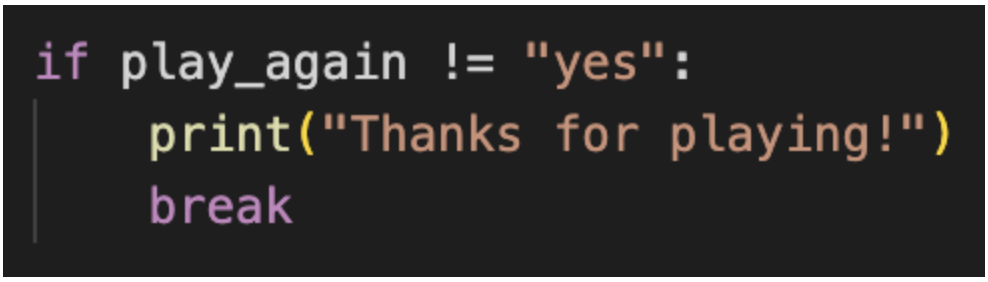
After updating the score, we print out the current score so the player can see who is winning. This can be done using a print statement like this:

```
print("Score: You =", player_score, "| Computer =", computer_score)
```

Finally, at the end of each round, we ask the player whether they want to play again. We do this by using the **input()** function:

```
play_again = input("Do you want to play again? (yes/no): ")
```

To check if the player wants to stop, we use the **!=** operator, which means "**not equal to**." So if the value of **play_again** is not equal to "**yes**", we print a thank-you message and use the break statement to exit the loop and end the game:



This is how you create a game that keeps running until the player decides to quit! Now our rock, paper scissors game is complete! Type **python rps.py** in the terminal.

Day 1 - Wrap up and dismiss

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 2

Class Notes

- Do quick reviews of the software when you open them up. Make sure the kids actually know how to use the programs independently and comfortably.

Day 2 Schedule

9:00 - 10:30 (9:00 - 10:15)	Word Guessing Game: Functions
10:30 - 11:30 (10:15 - 11:15)	**Morning Break**
11:30 - 12:30 (11:15 - 12:00)	Word Guessing Game: For Loop
12:30 - 1:15 (12:00 - 12:45)	**Lunch Break**
1:15 - 2:15 (12:45 - 1:30)	Word Guessing Game:: Colour and Random Selection
2:15 - 3:00 (1:30 - 2:15)	**Afternoon Break**
3:00 - 4:00 (2:15 - 3:00)	How to use Inkscape!
4:00 (3:00)	**Wrap Up and Dismiss**

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Start by creating a new file called **word_yourByteCampName.py**. We're going to build a new game where we first try to guess a single word, and eventually we'll guess from a random list of words.

Let's begin by creating a variable to store their target word we want the player to guess:

target = "byte"

Now, just like in Rock Paper Scissors, we're going to use the built-in input() function to let the player make a guess:

guess = input("Guess the word\n")

Try running the program to see if you can enter your first guess. Just like yesterday, type the command:

python3 ww_yourByteCampName.py

Reminder/Teaching Tip: After you've entered the terminal command once, you can bring it back by pressing the up arrow on your keyboard, finding it, and then pressing Enter to run it again.

The first rule for checking if a guess is valid is making sure it has the right number of letters. We'll do this using a basic if statement and Python's **built-in len() function**. Use the greater than **(>)** and less than **(<)** operators to let the player know if their guess is too long or too short compared to the target word.

```
guess_len = len(guess)

if guess_len > len(target):
    print("Your word contains too many letters!")
elif guess_len < len(target):
    print ("You word contains too few letters!")
else:
    # this means you have the right amount of letters
```

Next, let's add a win condition. If the player guesses the correct word, we'll print a victory message to the terminal. Use an if statement to check if the guess matches the target word. We'll place this check under the else clause since we'll be adding more conditions later and won't use elif for now.

if guess == target:
    print("You have guessed the word! You win!")

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Now let's make the game more interesting by giving the player feedback about which letters are correct and in the right position.

To do this, we'll introduce functions in Python. Explain that a function is a set of instructions that we can define once and call whenever we want the computer to run those steps.

Let's create the function call within the else branch to check the player's guess and the function definition right below our target variable. We will create a new variable called result to catch what's returned by the function which will be important later!

```
def check_guess(guess, target):
    # we will write our check_guess function here

guess = input("Guess the word\n")

guess_len = len(guess)

if guess_len > len(target):
    print("Your word contains too many letters!")
elif guess_len < len(target):
    print ("You word contains too few letters!")
else:
    # this means you have the right amount of letters

    result = check_guess(guess, target)

    if guess == target:
        print("You win!")
```

Day 2 - Morning Break

Day 2 - Block 2 - For Loop

Now that we have a guess, we're going to check if it matches the target word by comparing each letter individually using if statements.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Let's begin with the first letter. We need to ask a few things. First, is the letter in the correct spot? If it's not in the correct spot, is it somewhere else in the word? And if that isn't true either, then the letter is not in the word at all.

To build this logic, we'll use a few if statements. To make our messages clearer, we're going to introduce a new way of printing in Python, formatted printing. This allows us to include specific letters from the guess in our print messages.

To compare the first letter of the guess with the target, we'll use indexing. In Python, the index for the first letter is 0, so we'll use guess[0] and compare it to target[0].

```
if guess[0] == target[0]:
    print(f"{guess[0]} is in the correct spot.")
```

If the letter isn't in the right position, we then check whether it appears elsewhere in the word using the in keyword.

```
elif guess[0] in target:
    print(f"{guess[0]} is in the word but in the wrong spot.")
```

And finally, if neither of those checks is true, we can say that the letter is not in the word at all.

```
else:
    print(f"{guess[0]} is not in the word.")
```

Now that we've done this for the first letter, run the program in the terminal. Notice that it correctly checks whether the first letter is in the right spot or in the word. Next, let's do the same for the rest of the word. To do this, we need to loop through each letter.

Start this block by introducing for loops by writing an if statement up on the whiteboard. Feel free to make your own!

```
if having_fun:
    clap twice
else:
    clap once
```

```
for camper in bytecamp:
    if having_fun:
        clap twice
    else:
        clap once
```

Begin by demonstrating how to write an if statement that applies to just a single camper. Introduce a for loop that allows the program to repeat that same if statement for every camper in the group. Explain that by using a loop, the program can now evaluate each camper one at a time and apply the same logic or instructions to each of them individually.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Now we will use a for loop to loop through each letter of the word. To do that, we'll use a range based on the number of letters in the word. Inside the loop, we'll replace each 0 we used earlier with i to represent the current index of the word the program is on.

```
def check_guess(guess, target):
    for i in range(guess_len):
        if guess[i] == target[i]:
            print(f"{guess[i]} is in the correct spot.")
        elif guess[i] in target:
            print(f"{guess[i]} is in the word but in the wrong spot.")
        else:
            print(f"{guess[i]} is not in the word.")
```

Now just like yesterday we want to be able to keep guessing the word until we get it. Let's add a while loop to the main game functionality and a break statement if we win just like yesterday.

```
while True:
    guess = input("Guess the word\n")

    guess_len = len(guess)

    if guess_len > len(target):
        print("Your word contains too many letters!")
    elif guess_len < len(target):
        print ("You word contains too few letters!")
    else:
        # this means you have the right amount of letters
        result = check_guess(guess, target)
        if guess == target:
            print("You win!")
            break
```

In addition, now that we've learned how to use formatted print statements, let's make the game a bit easier by replacing our greater than and less than comparisons with a single != comparison.

```
if guess_len != len(target):
    print(f"Your word must contain {len(target)} letters")
```

Day 2 - Lunch Break

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 2 - Block 3 - Colour + Randomize

Now we're going to make our game much more fun by changing the color of the letters in our guess, depending on whether each letter is in the correct position, in the word but in the wrong spot, or not in the word at all. We'll do this by rebuilding the word inside our for loop and adding the appropriate ANSI color codes.

```
GREEN = "\033[92m"
YELLOW = "\033[93m"
RED = "\033[31m"
RESET = "\033[0m"
```

To make this work, we need to use accurate ANSI colour codes. These must be written exactly as shown. Copy them down and place them right above the target variable in your Python file.

I recommend projecting the codes onto the screen or writing them on a whiteboard so all the campers can write them down. These codes will go in front of each letter to control the font colour when it is displayed in the terminal.

Next, we will create a new variable called result in the check_guess function. This will start as an empty string. Add this at the beginning of the function, and add a return statement at the end so that the completed string can be sent back to the main part of the program.

Now let's build the result string.

First, we want to add the green colour code to any letter that is in the correct spot. We can combine strings using the plus operator and add each new piece onto the result using +=.

```
if guess[i] == target[i]:
    result += GREEN + guess[i] + RESET
```

Next, if the letter is in the word but not in the right spot, we will add the yellow colour code.

```
elif guess[i] in target:
    result += YELLOW + guess[i] + RESET
```

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Finally, if the letter is not in the word at all, we will add the red colour code.

```
else:
    result += RED + guess[i] + RESET
```

After we have gone through every letter, we can print the result string in the main game loop. If everything works, we will now see a beautifully coloured version of the guessed word in the terminal.

```
result = check_guess(guess, target)
print(result)
```

```
Guess the word
byqt
byqt
```

Now we want to be able to create a list of words that the computer can randomly choose from to set the target word.

Just like yesterday, let's start by importing random at the top of the file. Then, have the campers create a list of words like this:

word_list = ['bytecamp', 'ilovepython', 'metro', 'dodgeball']

Make sure the list contains strings, each separated by a comma, and that the campers include around 8 to 10 words of their choice.

Next, we'll use the random module to choose one word from the list as the target word:

target = random.choice(word_list)

Perfect! If there's extra time, you can add a limit to the number of guesses a player can make. To do this, create a variable called attempts at the top of the file and set it to a number like 6. Then, inside your game loop subtract 1 from the attempts variable at the end of the while loop after all the logic. You'll also want to change the condition of the while loop so that it runs only while attempts >= 1.

To help the player, add a print statement that shows how many attempts are left after you decrease the attempts. Finally, outside of the while loop, add a print that lets the player know they've lost the game if attempts reach 0.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

```
while attempts >= 1:
    guess = input("Guess the word\n")

    guess_len = len(guess)

    if guess_len != len(target):
        print(f"Your word must contain {len(target)} letters")
    else:
        result = check_guess(guess, target)
        print(result)
        if guess == target:
            print("You win!")
            break
    attempts -= 1
    print(f"You have {attempts} attempts left!")

if attempts == 0:
    print("you lose!")
```

Day 2 - Afternoon Break

Day 2 - Block 4 - Inkscape Title Scene

Start by creating a new folder Adventure-YourByteCampName, this is where you will store all your game assets and code. Create a folder called art within the folder.

> **NOTE: The folder must be named 'art' or the image to ascii generator will not work.**

Copy over both **game_template.py** and **image.py** from the Byte Camp Stuff folder.

In this block, we'll focus on creating a title scene. Begin by reviewing how to use Inkscape, then move on to building a game asset. Refer to the Inkscape Guide for instructions on how to teach the different tools.

Start with a basic warm-up file to help the kids get familiar with the tools. Once they're ready, have them create an Inkscape file for the title scene of their choice, with the document settings set to **100px by 100px**.

After you've created your own demo, show the class how the ASCII image generator works.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

It's very important to emphasize that the best images are made using **high-contrast colours, large shapes, and by fully utilizing the entire document space.**

INKSCAPE NOTE: Export the images as a jpg in order to get the best generated effect.

Day 2 - Wrap up and dismiss

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 3

Class Notes

- **VERY IMPORTANT**: Kids WILL be waiting for your help quite a bit over the next few days. A GOOD strategy is to have them work on easier tasks they won't need your help with. It will keep them from sitting around whining and doing nothing until you come around.
- **ALSO VERY IMPORTANT:** This is generally the hardest day (and most rewarding) for the kids. You MUST remain encouraging, positive, patient and energetic through the day. This is the most difficult part of the camp for you too!
- Give the kids small checkpoints to ensure certain assets are ready for reviews, or just simply to keep them on track.

Day 3 Schedule

9:00 - 10:30 (9:00 - 10:15)	Storyboarding CYOA game
10:30 - 11:30 (10:15 - 11:15)	**Morning Break**
11:30 - 12:30 (11:15 - 12:00)	Create all images!
12:30 - 1:15 (12:00 - 12:45)	**Lunch Break**
1:15 - 2:15 (12:45 - 1:30)	How to make a scene!
2:15 - 3:00 (1:30 - 2:15)	**Afternoon Break**
3:00 - 4:00 (2:15 - 3:00)	Backstory scene
4:00 (3:00)	**Wrap Up and Dismiss**

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 3 - Block 1 - Storyboarding + Inkscape

Here, the focus is on creating nine distinct scenes. The main goal is to illustrate the decisions they make through the images they create. It's important that each square represents a clear yes/no or either/or decision. Arrows should be drawn from each square, with "yes" or "no" written between the arrows to represent the path forward. They're also allowed to interleave decisions if they wish.

Once they have finished storyboarding their game they will begin creating their images + title image in Inkscape.

> **NOTE: You can also recommend the campers to create a specific end game scene, so that if the game has an exact path they want to follow. You can have them create an scene, that every wrong decision points towards.**

Day 3 - Morning Break

Day 3 - Block 2 - Inkscape Images cont.

Spend the rest of this block finishing the 9 images required for their game.

Day 3 - Lunch Break

Day 3 - Block 3 - Scene Making

After the images are made you can start by running the python script that will process all images files in the **art folder** into text files. You first have to create a virtual environment:

python3 -m venv venv

Now we need to install a python module to use our script to turn images into ascii:

source venv/bin/activate && pip install ascii_magic

Let's start by renaming the game_template file to **game_yourByteCampName.py.**

Let's have a look at making a scene. Every scene requires a couple key components in python:

- display_img('image_name.txt')
- print('') - prints text into the terminal
- input() - allows the user to put in an input into the screen
- path_function() - transitions the scene to a new scene (calls another function)

> **NOTE: When input is not assigned to a variable/if statement, any input put into the terminal will execute the next line in the function.**

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Let's create the title scene:

```
def start():
    display_img('title.txt')
    print("\nWelcome to the Game! Would you like to play?")
    input("Press Enter to start...")
    scene_one()
```

IMPORTANT: '/n' when written in a string starts a new line. Have the campers use this to make their games more user friendly.

Now that we have a scene lets practice running the game. Lets open the built in terminal in vscode by pressing the **ctrl +** ' button.

Type **python3 game_yourByteCampName.py** in the terminal. This will execute the game in the terminal.

IMPORTANT: you may only run the game in the virtual python environment. The virtual environment is activated in the terminal when '(venv)' appears at the beginning of the line. To reactivate it use 'source venv/bin/activate '.

You will notice that the game ends after putting in any input; this is because we have not defined a scene_one.

Now we will setup our first decision, the campers will create the first scene in their storyboard to do so.

```
def scene_one():
    display_img("gandalf.txt")
    print("\nGandalf asks you if you will go on an adventure. Will you go?")
    choice = input("Type 'yes' or 'no'\n")
```

Now that we have a choice, we will set up an if statement to make that choice.

```
def scene_one():
    display_img("gandalf.txt")
    print("\nGandalf asks you if you will go on an adventure. Will you go?")
    choice = input("Type 'yes' or 'no'\n")

    if choice == "yes":
        scene_two()
    elif choice == "no":
        end_scene()
    else:
        invalid_input(scene_one())
```

The invalid_input is a helper function designed to reprint the scene in case of any input not made in the if statement is chosen, however the scene function must be passed as a parameter.

If there is remaining time, have the campers start the next scenes.

Day 3 - Afternoon Break

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 3 - Block 4 - BackStory Scene

Now we're going to show how to add a text based backstory to a scene.

Start by creating a new list that contains each line that is wanted to be viewed separately as elements in the backstory.

```
def scene_one():
    backstory_one = [
        "The wind rustles through the trees... \n",
        "You hear faint footsteps in the distance...\n",
        "A shadowy figure approaches—it's Gandalf!\n"
    ]
```

Now use a for loop to display the story by pairing each element with a input statement wait for players input after they've read the backstory line.

```
for line in backstory_one:
    print(line)
    input("Press Enter to continue...")
```

After they've added backstory to one of their scenes, have them continue to work on their game!

Day 3 - Wrap up and dismiss

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 4

Class Notes

- **VERY IMPORTANT:** Kids WILL be waiting for your help quite a bit. A GOOD strategy is to work on the things they don't need as much help with, like title screens, character details, etc. Remind them!
- The first morning session is the best chance to get stuff done. Their brains are fresh and energy levels are high. Use it!!
- Some kids won't ask for much help. It does not mean that you don't have to spend much time with them. Aim to spend equal amount of time with everyone. Ask the quiet kids how things are going , suggest ideas/improvements.
- Your energy and teaching skills will affect how many kids will want to program! Do your best!

Day 4 Schedule

9:00 - 10:30 (9:00 - 10:15)	Pick ups
10:30 - 11:30 (10:15 - 11:15)	**Morning Break**
11:30 - 12:30 (11:15 - 12:00)	Inventory
12:30 - 1:15 (12:00 - 12:45)	**Lunch Break**
1:15 - 2:15 (12:45 - 1:30)	CYOA
2:15 - 3:00 (1:30 - 2:15)	**Afternoon Break**
3:00 - 4:00 (2:15 - 3:00)	CYOA
4:00 (3:00)	**Wrap Up and Dismiss**

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 4 - Block 1 - Pick Ups

Have the kids continue to add to their

Day 4 - Morning Break

Day 4 - Block 2 - Inventory

Day 4 - Lunch Break

Day 4 - Block 3 - CYOA

Day 4 - Afternoon Break

Day 4 - Block 4 - CYOA

Day 4 - Wrap up and dismiss

Just like the previous days, leave the laptops on for the parents to come in and take a look.
Don't forget to pat the kids on the back and congratulate them on a job well done!
Celebrating the small things really do wonders in boosting the students' confidence so make sure to acknowledge their growth and success!

Make sure to congratulate all of your campers on a job well done and to keep up the momentum for tomorrow!

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 5

Class Notes

- Mostly the same as Day 4
- Start assigning their QR codes if they haven't been done yet
- Check to make sure you have access to the upload folder for your camp. Don't wait until the last minute as the tech manager might not see the access request in time.
- Aim to be 100% done by lunch
- Collect their games before the end of 3rd block so you're ready to start uploads during block 4.
- Remember, all instructors should be outdoors with campers for breaks when possible. If one must stay indoors to resolve critical issues, consult with managers first.

Day 5 Schedule

9:00 - 10:30 (9:00 - 10:15)	Work on CYOA
10:30 - 11:30 (10:15 - 11:15)	**Morning Break**
11:30 - 12:30 (11:15 - 12:00)	Finish CYOA
12:30 - 1:15 (12:00 - 12:45)	**Lunch Break**
1:15 - 2:15 (12:45 - 1:30)	Export Games
2:15 - 3:00 (1:30 - 2:15)	**Afternoon Break**
3:00 - 4:00 (2:15 - 3:00)	Play eachother's games
4:00 (3:00)	**Wrap Up and Dismiss**

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Day 5 - Block 1 -

Final push to get all things done!

REMEMBER: Periodically check in on the kids. They don't need to be asking questions in order to get attention from you, get down to their level and ask for a progress report/ demo of their project.

Day 5 - Morning Break

Day 5 - Block 2 -

Day 5 - Lunch Break

Day 5 - Block 3 -

This is it. This is the last block to put the final touches on their projects because before the final break time starts, their folders and projects are to be collected for the Master USB, ready for uploading as soon as you return.

Day 5 - Afternoon Break

Final Cut Off. No camper is permitted to stay indoors to finish and all instructors are to go outside with the campers. For Upload Instructions, check your attendance sheet for the guide link.

Day 5 - Block 4 - Open Arcade/Movie Showcase

While one instructor starts running the uploads, another instructor can lead the kids through completing feedback surveys.
Once the surveys are done and completed, students may spend the remainder of the block admiring their classmates' projects.

About halfway through this block, hand out the QR codes and explain how they work.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

Hopefully the projects are also done early enough that you are able to open the classroom 15 minutes early and parents may come in to also admire and enjoy the projects made in the camp.

//You can create a VLC playlist of all the students' movies and have them on loop so as parents are coming and going, they will be able to watch the movies while the students are packing themselves up.

Day 5 - Wrap up and dismiss

Refer to the "**Friday Finishes**" section of the **"Weekly Reminders"** if you don't recall how to wrap up the camp.
SUPER IMPORTANT! THAT'S WHY IT'S IN LARGE LETTERS!

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.

© Copyright Byte Camp 2025 - Materials on this page are permitted to be used and reproduced by Byte Camp Education Society only.