```
 .----------------.
| .--------------. |
| |   ______     | |
| |  |_   __ \   | |
| |    | |__) |  | |
| |    |  ___/   | |
| |   _| |_      | |
| |  |_____|     | |
| |              | |
| '--------------' |
 '----------------'
 .----------------.
| .--------------. |
| |  ______      | |
| | |_   ___  | | |
| |   | |_  \_|  | |
| |   |  _|      | |
| |  _| |_       | |
| | |_____|      | |
| |              | |
| '--------------' |
 '----------------'
 .----------------.
| .--------------. |
| |   _____      | |
| |  |_   _|     | |
| |    | |       | |
| |    | |   _   | |
| |   _| |__/ |  | |
| |  |________|  | |
| |              | |
| '--------------' |
 '----------------'
```

PYTHON FOR LOSERS

Chapter 1: Welcoming Again – Toaster 💀

Welcome back, you absolute loser 🤡. If you’re here, that means you either survived Python for Losers (Book 1) or you just randomly picked up this book thinking it’s gonna turn you into a tech god overnight. Spoiler alert: it won’t 💀.

But don’t worry, I’m here to roast you till you’re crispy golden brown and somehow, in the process, you’ll learn some Python too. Win-win, right?

What’s New in This Book?

In the first book, you probably wrote some useless print statements, played around with variables like a caveman discovering fire, and maybe—just maybe—made a tiny “Hello World” script thinking you’re the next Elon Musk.

But this time? We go bigger.
We’re gonna do some actual coding that doesn’t look like a 5-year-old smashing a keyboard.

What You Need Before Starting:

✅ A brain (or at least 2% of it functioning)
✅ Python installed (if not, Google it, I'm not your dad)
✅ Some patience (because debugging is just crying in a cooler way)
✅ A snack, because let's be real, you'll be here for a while

First Useless Code to Get Started:

Alright, let's warm up. Open Python and type this masterpiece:

```
print("Welcome back, loser! Ready to suffer again?")
```

Now run it. If it works, congrats—you're still a loser but at least one with working Python. If it doesn't, well… skill issue 💀.

What's Next?

In the next chapter, we'll actually start coding like a slightly less useless person. Until then, enjoy the roast and try not to burn your CPU in frustration. 💀🔥

End of Chapter 1

Chapter 2: Variables – Stop Naming Them 'x' and 'y', You Caveman 💀

Alright, loser, you made it to Chapter 2. That means either you actually want to learn Python or you're just here to get roasted for fun. Either way, I don't care, because we're diving into something important but boring as hell – variables.

What the Hell Are Variables?

Think of a variable as a box where you can store stuff. You give the box a name, throw some value into it, and later, you can open it up and use whatever's inside.

Example:

You put money in a wallet → wallet = money

You put your last two brain cells in a box → box = "last two brain cells"

You put all your hopes and dreams into Python → dreams = "gone"

In Python, this looks like:

```
wallet = 100
box = "last two brain cells"
dreams = "gone"

print(wallet)  # This prints: 100
print(box)  # This prints: last two brain cells
print(dreams)  # This prints: gone
```

If you don't understand this yet, don't worry, your brain just needs a firmware update.

Rules for Naming Variables (That You'll Ignore Anyway 🤡)

1. Names must start with a letter or underscore (_) – No numbers first, you caveman.

```
_private = "hidden"  # ✅ Works
3am_thoughts = "why am I alive?"  # ❌ ERROR: Can't start with a number
```

2. No spaces – Python ain't a fan of gaps in variable names.

```
my name = "Bob"  # ❌ ERROR
my_name = "Bob"  # ✅ Works
```

3. Be descriptive, not a dumbass

```
x = 5  # ❌ What is x? The number of brain cells left?
num_apples = 5  # ✅ Makes sense
```

Types of Variables (Because Python Ain't Psychic)

Python automatically detects the type of data, but let's be real—you should at least know what's happening.

```
age = 15  # Integer (whole number)
price = 99.99  # Float (decimal number)
name = "Dwip"  # String (text)
is_alive = True  # Boolean (True or False)
```

You can check the type of any variable using type():

```
print(type(age))  # <class 'int'>
print(type(price))  # <class 'float'>
print(type(name))  # <class 'str'>
print(type(is_alive))  # <class 'bool'>
```

If you expected something cooler, too bad.

What Happens If You Mess Up? (Which You Will 💀)

Try this:

```
num = "10"
print(num + 5)  # ERROR: You can't mix strings and numbers
```

Boom. Python screams at you because "10" is text, not a number. Fix it with:

```
num = int("10")  # Convert string to number
print(num + 5)  # ✅ Works fine
```

Final Useless Task Before the Next Chapter

Alright, since you survived this chapter, type this:

```
name = input("Enter your name, loser: ")
age = input("How old are you?: ")
print("Damn,", name, "you're already", age, "years old? Time flies, huh?")
```

Run it, enter your name and age, and feel old. That's your punishment for reaching Chapter 2. 💀

Next Chapter?

We'll get into operators, aka how to make Python do math for you so you don't have to use your last two neurons. Until then, don't name your variables x and y like a prehistoric ape.

Chapter 2: Operators – Making Python Do Your Homework While You Sleep 💀

Alright, listen up, nerd. Now that you know how to name variables (hopefully not just 'x' and 'y' like a caveman), it's time to make Python do actual work.

Today, we talk about operators, aka how to make Python do basic math, compare shit, and manipulate numbers like a pro.

1. Arithmetic Operators – Your Math Teacher's Worst Nightmare

Python is basically a free calculator that won't judge your bad math skills. It can:

+ → Addition (5 + 2 → 7)

- → Subtraction (5 - 2 → 3)

* → Multiplication (5 * 2 → 10)

/ → Division (5 / 2 → 2.5) (always gives decimal)

// → Floor Division (5 // 2 → 2) (chops off decimals like your dad chopped your gaming hours)

% → Modulus (5 % 2 → 1) (gives remainder, perfect for guessing odd/even numbers)

** → Exponent (5 ** 2 → 25) (because sometimes you want to go full Einstein mode)

💀 Example:

```
a = 10
b = 3

print("Addition:", a + b)  # 13
print("Subtraction:", a - b)  # 7
print("Multiplication:", a * b)  # 30
print("Division:", a / b)  # 3.3333
print("Floor Division:", a // b)  # 3
print("Remainder:", a % b)  # 1
print("Power:", a ** b)  # 1000
```

2. Comparison Operators – Let's Compare Shit Like Relatives Compare You to Sharmaji's Kid

Python can compare numbers like your parents compare your marks 💀.

== → Equal to (5 == 5 → True) (checks if both values are the same)

!= → Not equal to (5 != 3 → True) (opposite of ==, for when you wanna reject reality)

> → Greater than (5 > 3 → True) (flexing who's bigger)

< → Less than (5 < 3 → False) (loser behavior)

>= → Greater than or equal to (5 >= 5 → True) (half-flexing)

<= → Less than or equal to (5 <= 3 → False) (just accept defeat)

💀 Example:

```
marks = 69
passing_marks = 40

if marks >= passing_marks:
    print("You passed, legend!")
else:
    print("Go study, dumbass!")
```

3. Logical Operators – How to Make Python Think Like a Procrastinator

These are used when you want multiple conditions at once, like checking if you failed in both Math and Science 💀.

and → Both conditions must be True (True and False → False)

or → At least one condition must be True (True or False → True)

not → Flips the value (not True → False) (because Python likes gaslighting itself)

💀 Example:

```
maths = 20
science = 35

if maths < 40 and science < 40:
    print("Double fail, bro 💀")
elif maths < 40 or science < 40:
    print("At least you passed one subject 🤡")
else:
    print("Genius mode activated!")
```

4. Assignment Operators – The Lazy Way to Update Variables

Instead of writing x = x + 5 like a noob, Python lets you do it in one line.

= → Assign (x = 5) (normal assignment)

+= → Add and assign (x += 5 same as x = x + 5)

-= → Subtract and assign (x -= 5)

*= → Multiply and assign (x *= 5)

/= → Divide and assign (x /= 5)

%= → Modulus and assign (x %= 5)

**= → Exponent and assign (x **= 2)

//= → Floor divide and assign (x //= 5)

💀 Example:

```
x = 10
x += 5  # Same as x = x + 5
print(x)  # 15

x *= 2  # Same as x = x * 2
print(x)  # 30
```

5. Bitwise Operators – For When You Want to Feel Like a Hacker (But Probably Won't Use These)

These work at the binary level and are mostly used in low-level programming. But since you'll see them at some point, here they are:

& → AND (both bits must be 1 to give 1)

| → OR (at least one bit must be 1)

^ → XOR (only one bit can be 1, not both)

~ → NOT (flips all bits)

<< → Left Shift (shifts bits left, multiplying by 2)

>> → Right Shift (shifts bits right, dividing by 2)

💀 Example:

```
a = 5  # Binary: 0101
b = 3  # Binary: 0011

print(a & b)  # 1  (0001)
print(a | b)  # 7  (0111)
print(a ^ b)  # 6  (0110)
print(~a)  # -6  (inverts bits)
print(a << 1)  # 10 (shifts left, multiplies by 2)
print(a >> 1)  # 2  (shifts right, divides by 2)
```

(Yeah, I know, you won't use this until you're coding some ultra-optimized bullshit, but still, now you know.)

Final Words – What Did You Learn, You Lazy Coder?

Arithmetic Operators → Basic math, but cooler.

Comparison Operators → Helps Python judge numbers like your relatives judge your career.

Logical Operators → Makes Python think logically (unlike you).

Assignment Operators → Saves typing effort, because you're lazy.

Bitwise Operators → Makes you look like a hacker (but you'll never use them).

Now go practice before you forget everything in 5 minutes.

Chapter 3: If-Else – Teaching Python How to Judge You Like Your Relatives 💀

Alright, dumbass, now that you can do basic math in Python (hopefully without embarrassing yourself), it's time to make Python think.

You see, a program that just spits out numbers like a calculator is lame as hell. We need it to make decisions, like:

Should you pass a test or get roasted by your parents?

Did you get enough sleep or are you running on 2 brain cells?

Are you broke as hell or just pretending?

That's where if-else statements come in. These let Python judge stuff and take action—kind of like how society does, but with less disappointment.

1. If-Else Basics – Python, Meet Logic (Something You Lack 💀)

The basic format looks like this:

```
if condition:
    # do something
else:
    # do something else
```

Python reads this top to bottom, checking if the condition is true. If it is, it runs the first block. If not, it runs the else block. Simple, right?

💀 Example:

```
marks = 35

if marks >= 40:
    print("You passed! 🎉")
else:
    print("Go study, dumbass! 📚")
```

Output:

```
Go study, dumbass! 📚
```

😂 Tough luck, loser. Maybe next time, try studying instead of scrolling 🟧⬛ at 2 AM.

2. Elif – Because Life Ain't Just Black and White

Sometimes, you need more than just two choices. Enter elif (short for "else if"), which gives Python multiple options to check.

💀 Example:

```
marks = 69  # Nice

if marks >= 90:
    print("Bro, you're a genius! 🔥")
elif marks >= 60:
    print("Not bad, but try harder. 🤓")
elif marks >= 40:
    print("At least you passed... barely. 😬")
else:
    print("Go become a TikToker or something. 💀")
```

Output:

```
Not bad, but try harder. 🤓
```

💀 Translation: You're not dumb, but you're not smart either. You're just... there.

3. Nested If – Because Life Gets Complicated

What if you need to check multiple conditions at once? That's where nested if statements come in.

💀 Example:

```
age = 18
money = 10

if age >= 18:
    if money >= 100:
        print("You can enter the club! 🍾")
    else:
        print("Bruh, you're broke. Stay outside. 💀")
else:
    print("Go home, kid. 🤡")
```

Output:

```
Bruh, you're broke. Stay outside. 💀
```

😂 Translation: You're old enough, but you forgot to bring money, you clown. Better luck next time.

4. Short If – For Lazy People (Like You)

If you only have one line to execute, Python lets you shorten the if-else statement.

💀 Example:

```
marks = 85
print("Passed! 🎉") if marks >= 40 else print("Failed. 💀")
```

Output:

```
Passed! 🎉
```

Why write 5 lines when 1 is enough? Save time so you can waste it on YouTube Shorts instead.

5. Logical Operators in If-Else – Making Python Think Like a Human (Almost)

Sometimes you need to combine conditions because life ain't that simple.

💀 Example:

```
age = 22
has_ID = True

if age >= 18 and has_ID:
    print("Welcome to the club! 🍹")
else:
    print("Go home, kid. 🤡")
```

Translation: No ID = No entry. Even if you're 100 years old, bouncer doesn't give a damn.

6. The Ultimate If-Else Roasting Machine 🔥

Let's make Python roast you based on your life choices 💀.

💀 Example:

```
hours_of_sleep = int(input("How many hours did you sleep? "))

if hours_of_sleep >= 8:
    print("Well rested and productive! 🔥")
elif hours_of_sleep >= 5:
    print("You're surviving... barely. 😬")
elif hours_of_sleep >= 3:
    print("Bro, you're running on fumes. 💀")
else:
    print("Go to bed, dumbass. You're gonna die. 💀💀💀")
```

🤡 Test it on yourself and cry.

Final Words – What Did We Learn Today?

if → Checks a condition

else → Runs if all else fails (like your exam prep 💀)

elif → Adds more options (because life is unfair)

and, or, not → Helps Python think better than you

Nested if → For when life gets complicated

Homework (LOL, Just Try This and Don't Cry 💀)

Try making Python judge your financial situation based on how much money you have. 💀

```
money = int(input("How much money do you have? "))

if money >= 1000:
    print("Rich kid, buy me something! 🤑")
elif money >= 500:
    print("Decent, but not flex-worthy. 😎")
elif money >= 100:
    print("Okay, you’re surviving. 🤡")
else:
    print("Broke as hell 💀 Go touch grass.")
```

💀 Try it, then question your life decisions.

Chapter 4: Loops – Making Python Repeat Your Failures Until It Crashes 💀

Alright, dumbass, so now Python can judge you like your parents using if-else. But what if you want it to repeat something multiple times? Like:

Retrying JEE until you finally give up and become a YouTuber 💀

Scrolling reels like a brain-dead monkey until your battery dies

Spamming "Hi" to your crush even though she left you on seen 💀

That’s where loops come in. Python can repeat stuff automatically so you don’t have to.

1. The While Loop – When You Don’t Know When to Stop (Like You on 🟧⬛ at 2 AM 💀)

A while loop runs until a condition becomes false. In other words, it keeps running until you tell it to stop—or until Python crashes trying to process your stupidity.

💀 Example:

```
x = 1

while x <= 5:
    print(f"Attempt {x}: Still a loser 💀")
    x += 1
```

Output:

```
Attempt 1: Still a loser 💀
Attempt 2: Still a loser 💀
Attempt 3: Still a loser 💀
Attempt 4: Still a loser 💀
Attempt 5: Still a loser 💀
```

😂 Translation: Python will keep roasting you until x becomes 6.

⚠️ Warning: Infinite Loops = RIP Your PC 💀

If you forget to update the variable inside the loop, Python will never stop running.

💀 Example of a mistake:

```
while True:
    print("Help! I'm stuck in an infinite loop! 💀")
```

This will NEVER STOP unless you manually close Python or throw your PC out the window.

😂 Moral of the story: Don't write dumb loops unless you want your PC to overheat.

2. The For Loop – Python's Way of Saying "Do This X Times and Move On"

Unlike while, a for loop knows exactly how many times to run. It's best for counting stuff, looping through lists, or automating dumb tasks.

💀 Example:

```
for i in range(5):
    print(f"Try {i+1}: Still broke 💀")
```

Output:

```
Try 1: Still broke 💀
Try 2: Still broke 💀
Try 3: Still broke 💀
Try 4: Still broke 💀
Try 5: Still broke 💀
```

😂 Python just summarized your financial situation.

3. The Range Function – Python's Built-in Calculator for Loops

range(start, stop, step) helps control how the loop counts.

💀 Examples:

```
for i in range(2, 10, 2):  # Start at 2, stop before 10, step by 2
    print(i)
```

Output:

```
2
4
6
8
```

😂 Translation: Python is counting like your IQ dropping every time you touch your phone.

4. Looping Through Lists – Python Becoming a Stalker

Loops aren't just for numbers—they can go through entire lists too.

💀 Example:

```
crushes = ["Emily", "Sophia", "Aditi", "Riya"]

for girl in crushes:
    print(f"Texting {girl}... ignored. 💀")
```

Output:

```
Texting Emily... ignored. 💀
Texting Sophia... ignored. 💀
Texting Aditi... ignored. 💀
Texting Riya... ignored. 💀
```

😂 Bruh, even Python can tell you're getting NO REPLIES.

5. Break and Continue – Controlling Loops Like a Manipulative Ex

Sometimes, you want to exit a loop early or skip certain parts. That's where break and continue come in.

💀 Break Example: (Exit the loop if something happens)

```
for money in range(100, 0, -10):
    print(f"Wallet: {money} left")

    if money <= 50:
        print("Broke as hell 💀")
        break
```

Output:

```
Wallet: 100 left
Wallet: 90 left
Wallet: 80 left
Wallet: 70 left
Wallet: 60 left
Wallet: 50 left
Broke as hell 💀
```

😂 Even Python knows when you're out of money.

💀 Continue Example: (Skip certain parts of a loop)

```
for i in range(1, 6):
    if i == 3:
        print("Skipping number 3 because it's cursed 💀")
        continue

    print(f"Number {i}")
```

Output:

```
Number 1
Number 2
Skipping number 3 because it's cursed 💀
Number 4
Number 5
```

😂 Python out here avoiding bad luck like you avoid responsibilities.

6. Nested Loops – Because One Isn't Enough (Like Your Crush's Boyfriends 💀)

A loop inside a loop can be useful for working with grids, tables, or just making Python suffer.

💀 Example:

```
for i in range(3):
    for j in range(3):
        print(f"({i}, {j})", end=" ")
    print()
```

Output:

```
(0, 0) (0, 1) (0, 2)
(1, 0) (1, 1) (1, 2)
(2, 0) (2, 1) (2, 2)
```

😂 Looks fancy, but don't worry—you'll probably never use this in real life.

Final Words – What Did We Learn Today?

while → Repeats forever unless stopped (like your stupidity) 💀

for → Runs a fixed number of times (like your internet before daily limit dies) 💀

break → Stops the loop early (like you stopping your gym plans) 💀

continue → Skips one loop iteration (like you skipping homework) 💀

range() → Controls loop numbers (so you don't have to count on fingers, dumbass) 💀

Nested loops → More loops = More pain for your PC 💀

Homework (If You Dare 💀)

Try making a countdown to your exams using a while loop.

```
days_left = int(input("Days left for exams: "))

while days_left > 0:
    print(f"{days_left} days left. Still not studying. 💀")
    days_left -= 1

print("Exam day! Good luck failing! 💀💀💀")
```

😂 Try it, then cry in a corner.

Chapter 5: Functions – Stop Copy-Pasting Like a Dumbass 💀

Alright, bitch, so now you can make Python loop and repeat stuff. But are you still copy-pasting the same code like an idiot? If so, this chapter is gonna slap some sense into you.

1. What the F*ck is a Function?

A function is like a reusable piece of code that you can call anytime. Instead of writing the same dumb shit again and again, you just define a function once and use it whenever needed.

💀 Example of Being a Dumbass (Without Functions):

```
print("Hello, dumbass!")
print("Hello, dumbass!")
print("Hello, dumbass!")
print("Hello, dumbass!")
print("Hello, dumbass!")
```

😂 Bruh, why tf are you repeating this five times?

💀 Now, Let's Use a Function Like a Smartass:

```
def roast():
    print("Hello, dumbass!")

roast()
roast()
roast()
roast()
roast()
```

😂 Same output, but cleaner than your search history.

2. Defining Functions – Tell Python WTF to Do

To create a function, you use def, which stands for "Define this f*cking function."

💀 Basic Function Syntax:

```
def function_name():
    # Some code
```

💀 Example:

```
def insult():
    print("Your brain has encountered an error 💀")

insult()
insult()
```

😂 Translation: You just made Python roast people on demand.

3. Functions with Arguments – Customize Your Roasts

Functions can also take input (called arguments) so they can do different things depending on what you pass.

💀 Example:

```
def roast(name):
    print(f"{name}, your IQ is dropping faster than Bitcoin 💀")

roast("Rahul")
roast("Neha")
roast("You")
```

Output:

```
Rahul, your IQ is dropping faster than Bitcoin 💀
Neha, your IQ is dropping faster than Bitcoin 💀
You, your IQ is dropping faster than Bitcoin 💀
```

😂 Now you can roast your friends dynamically.

4. Functions with Return – Getting Something Back Instead of Just Crying

Sometimes, a function should give you a result instead of just printing shit.

💀 Example:

```
def add(a, b):
    return a + b

result = add(5, 10)
print(f"Your dumb ass answer: {result}")
```

Output:

```
Your dumb ass answer: 15
```

😂 Now Python is smarter than you at maths.

5. Default Arguments – Lazy Mode Activated

You can set default values for function arguments, so if the user doesn't pass anything, Python won't crash.

💀 Example:

```
def greet(name="Bitch"):
    print(f"Hello, {name}, welcome to hell 💀")

greet()  # Uses default
greet("Sahil")  # Uses given name
```

Output:

```
Hello, Bitch, welcome to hell 💀
Hello, Sahil, welcome to hell 💀
```

😂 Python auto-fills the insult for you.

6. Multiple Arguments – More Ways to F*ck Up

You can pass any number of arguments using *args.

💀 Example:

```
def roast_all(*names):
    for name in names:
        print(f"{name}, you still single? Damn. 💀")

roast_all("Arjun", "Rohan", "Aditi")
```

Output:

```
Arjun, you still single? Damn. 💀
Rohan, you still single? Damn. 💀
Aditi, you still single? Damn. 💀
```

😂 Python just exposed your friend circle.

7. Lambda Functions – Shortcuts for Lazy Programmers

A lambda function is a one-line function for when you're too lazy to write def.

💀 Example:

```
double = lambda x: x * 2

print(double(5))  # Output: 10
```

😂 Same as a normal function, but written in lazy mode.

8. Recursion – When a Function Calls Itself Like a Dumbass

A recursive function calls itself until it reaches a stopping condition.

💀 Example:

```
def countdown(n):
    if n <= 0:
        print("Boom! 💥")
    else:
        print(n)
        countdown(n - 1)
countdown(5)
```

Output:

```
5
4
3
2
1
Boom! 💥
```

😂 Looks cool, but don't use this too much or Python will have a meltdown.

Final Words – What Did We F*cking Learn?

def → Creates a function, so you stop writing dumb repetitive code.

Arguments → Let functions do different things based on input.

return → Gives back a result instead of just printing sh*t.

Default values → So you can be lazy and avoid errors.

*args → Allows multiple inputs like a pro.

Lambda functions → Shortcut functions for ultra-lazy coders.

Recursion → A function calling itself like a dumbass.

Homework (If You're Not a P*ssy 💀)

Try making a function that calculates how many days you've wasted on reels.

```
def wasted_days(hours_per_day):
    return hours_per_day * 365

hours = int(input("Hours wasted per day: "))
print(f"You waste {wasted_days(hours)} hours a year, dumbass 💀")
```

😂 Try it, then go cry in a corner.

Chapter 6: Lists – Stop Hardcoding Like a Dumbass 💀

Alright, bitch, so now you know how to use functions. But are you still writing a separate variable for every piece of data? If so, I got bad news for you: You program like a dumbass. Time to fix that with lists.

1. What the F*ck is a List?

A list is basically a collection of items. Instead of making 100 variables, you put all that shit inside one list and access them like a pro.

💀 Example of Being a Dumbass (Without Lists):

```
item1 = "Burger"
item2 = "Pizza"
item3 = "Maggie"
item4 = "Coke"
```

😂 Bruh, why tf are you making so many variables?

💀 Now, Let’s Use a List Like a Smartass:

```
items = ["Burger", "Pizza", "Maggie", "Coke"]
print(items)
```

Output:

```
['Burger', 'Pizza', 'Maggie', 'Coke']
```

😂 Cleaner, shorter, and doesn’t make you look like a dumbass.

2. Accessing List Items – Don’t Be a Noob

You can access any item in a list using its index (position).
💀 Example:

```
food = ["Burger", "Pizza", "Maggie", "Coke"]
print(food[0])  # First item
print(food[2])  # Third item
```

Output:

```
Burger
Maggie
```

😂 Remember: Indexing starts from 0, not 1, dumbass.

3. Changing List Items – Customize Your Sh*t

You can replace any item in a list like this:

💀 Example:

```
food = ["Burger", "Pizza", "Maggie", "Coke"]
food[1] = "Fries"
print(food)
```

Output:

```
['Burger', 'Fries', 'Maggie', 'Coke']
```

😂 Now your list is customizable, unlike your exam scores.

4. Adding New Items – Because One is Never Enough

There are multiple ways to add new shit to a list:

💀 1. append() – Add an item to the end

```
food = ["Burger", "Pizza"]
food.append("Pasta")
print(food)
```

Output:

```
['Burger', 'Pizza', 'Pasta']
```

😂 Now you’re stacking food like your unfinished projects.

💀 2. insert() – Add an item at a specific position

```
food.insert(1, "Fries")
print(food)
```

Output:

```
['Burger', 'Fries', 'Pizza', 'Pasta']
```

😂 Put it wherever tf you want.

5. Removing Items – Delete That Useless Sh*t

You can remove items in different ways:

💀 1. remove() – Delete by value

```
food = ["Burger", "Pizza", "Maggie"]
food.remove("Pizza")
print(food)
```

Output:

```
['Burger', 'Maggie']
```

😂 Say goodbye to whatever tf you don't like.

💀 2. pop() – Delete by index (or last item by default)

```
food.pop(1)  # Removes item at index 1
print(food)
```

Output:

```
['Burger']
```

😂 Bye bye, Pizza.

6. Looping Through Lists – Don't Be a Repetitive Dumbass

Instead of accessing items one by one like a caveman, just loop through the damn list.

💀 Example:

```
food = ["Burger", "Pizza", "Maggie"]
for item in food:
    print(f"I'm eating {item} 😋")
```

Output:

```
I'm eating Burger 😋
I'm eating Pizza 😋
I'm eating Maggie 😋
```

😂 Now Python eats for you.

7. List Length – Find Out How F*cking Long It Is

Use len() to count the number of items in a list.

💀 Example:

```
food = ["Burger", "Pizza", "Maggie"]
print(len(food))
```

Output:

```
3
```

😂 Now you know exactly how much junk food you eat.

8. Sorting Lists – Put That Sh*t in Order

💀 1. sort() – Sort in ascending order

```
numbers = [4, 2, 8, 1, 6]
numbers.sort()
print(numbers)
```

Output:

```
[1, 2, 4, 6, 8]
```

😂 Now it’s in proper order, unlike your life.

💀 2. reverse() – Reverse the damn list

```
numbers.reverse()
print(numbers)
```

Output:

```
[8, 6, 4, 2, 1]
```

😂 Backwards, like your exam preparation.

9. Copying Lists – Stop F*cking Up References

If you copy a list like this:

```
a = [1, 2, 3]
b = a
```

You just f*cked up because both a and b now point to the same list. Change one, both change.

💀 Fix it using .copy():

```
a = [1, 2, 3]
b = a.copy()
b[0] = 100
print(a)
print(b)
```

Output:

```
[1, 2, 3]
[100, 2, 3]
```

😂 Now b is independent, unlike your bank account.

10. Nested Lists – Lists Inside Lists, Like Some Inception Sh*t

You can store lists inside other lists for advanced shit.

💀 Example:

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(matrix[1][2])  # Access second row, third item
```

Output:

6

😂 Matrix coding, but without getting into the Matrix.

Final Words – What Did We F*cking Learn?

Lists save your dumb ass from making 100 variables.

Use append(), insert(), remove(), pop() to modify lists.

Loop through lists like a smartass, don't hardcode.

Use sort() and reverse() to organize your mess.

Use .copy() if you don't wanna mess up references.

Nested lists exist, but don't overcomplicate your life.

Homework (If You're Not a P*ssy 💀)

Try making a list that stores your top 5 most watched 🟧⬛ video categories and then print them one by one.

💀 Example (Don't lie, we know what you watch):

```
categories = ["Thick", "POV", "Latina", "Stepsis", "ASMR"]
for category in categories:
    print(f"Your fav category: {category} 💀")
```

😂 Try it, and then go clear your browser history.

Chapter 7: Tuples – The "Look But Don't Touch" Type 💀

Alright, dumbass, we're moving on to something even simpler than lists. But this time, you can't f*ck with it. Introducing: Tuples.

1. What the F*ck is a Tuple?

A tuple is just like a list, but you can't change it.
Yeah, no append(), no remove(), no modifications.
It's like having a strict-ass teacher who won't let you change your answers after submitting the exam.

💀 Example of a Tuple:

```
foods = ("Burger", "Pizza", "Maggie", "Coke")
print(foods)
```

Output:

```
('Burger', 'Pizza', 'Maggie', 'Coke')
```

😂 Looks just like a list, right? But try to change it and Python will slap your ass.

2. Why the F*ck Use Tuples?

If you can't change them, why tf do we use them?

When you don’t want things to be accidentally modified.

When performance matters (tuples are faster than lists).

When you need data security (like passwords, constants, etc.).

😂 Think of tuples as your browser history. Once it’s saved, you can’t change it… but you can delete the whole thing.

3. Accessing Tuple Items – Same Sh*t as Lists

You access tuple items the same way as lists.

💀 Example:

```
foods = ("Burger", "Pizza", "Maggie", "Coke")
print(foods[0])  # First item
print(foods[2])  # Third item
```

Output:

```
Burger
Maggie
```

😂 See? Same as lists, but without the ability to change sh*t.

4. What Happens If You Try to Modify a Tuple? (Dumbass Alert 🚨)

💀 Example of f*cking up:

```
foods = ("Burger", "Pizza", "Maggie", "Coke")
foods[1] = "Fries"  # Trying to modify a tuple
```

🚨 Python will throw a f*cking error:

```
TypeError: 'tuple' object does not support item assignment
```

😂 Good luck explaining that to your teacher.

5. Tuple Methods – What Can You Actually Do?

Tuples are lazy as f*ck, so they don’t have many methods.
Here’s the only useful sh*t they can do:

💀 1. count() – Counts how many times an item appears.

```
nums = (1, 2, 2, 3, 4, 2)
print(nums.count(2))  # How many times does 2 appear?
```

Output:

```
3
```

😂 Bro, it’s just counting, that’s all it can do.

💀 2. index() – Finds the position of an item.

```
nums = (1, 2, 3, 4, 5)
print(nums.index(3))  # Where is 3?
```

Output:

2

😂 That's it. That's all tuples can do.

6. Converting Tuples to Lists – Because You'll Get Tired of This Sh*t

If you ever get sick of tuples being unchangeable, just convert them to a list.

💀 Example:

```
foods = ("Burger", "Pizza", "Maggie")
foods_list = list(foods)  # Convert to list
foods_list.append("Fries")  # Now we can modify
print(foods_list)
```

Output:

```
['Burger', 'Pizza', 'Maggie', 'Fries']
```

😂 Congratulations, you just f*cking cheated the system.

7. Deleting Tuples – Burn That Sh*t

You can't modify a tuple, but you can delete the whole damn thing.

💀 Example:

```
foods = ("Burger", "Pizza", "Maggie")
del foods  # Deletes the whole tuple
```

😂 Now it's gone, just like your motivation to study.

8. When to Use Tuples vs. Lists?

Here's a simple breakdown for your dumbass brain:

Can you modify it?

✅ Lists: Yes

❌ Tuples: No

Which one is faster?

❌ Lists: Slower

✅ Tuples: Faster

Which one is more secure?

❌ Lists: Nah, anyone can change it

✅ Tuples: Yes, no one can f*ck with it

Can you sort the elements?

✅ Lists: Yes

❌ Tuples: No

😂 So basically, if you need to modify sh*t, use a list. If not, use a tuple. Simple as that.

Final Words – What Did We F*cking Learn?

Tuples are like lists, but you can't change them.

They are faster, more secure, and good for constant data.

You can still access and loop through them.

Convert them to lists if you ever need to modify them.

Or just delete the whole thing if you regret your life choices.

Homework (If You Got Balls 💀)

Try making a tuple that stores your top 3 worst exam scores and print them.
(😂 We all know there's a 0 in there.)

💀 Example:

```
exam_scores = (20, 45, 0)
for score in exam_scores:
    print(f"Your sh*tty score: {score} 💀")
```

😂 Do it, and then go study, dumbass.

9. Dictionary – The Gangster of Python

Alright, bitches, it's time to meet dictionaries—the f*cking gangsters of Python. They don't mess around with indexes like lists; they use keys to store and retrieve data. Think of them as an OP storage unit that actually makes sense.

What the Hell is a Dictionary?

A dictionary in Python is like your phone's contact list. Instead of using numbers like a dumbass, you use names (keys) to find the info (values).

Here's how it looks:

```
gang_dict = {
    "name": "Dwip The Roaster",
    "age": 9999,
    "job": "Destroying Noobs"
}
print(gang_dict["name"])  # Output: Dwip The Roaster
```

Why Dictionaries Are OP as F*ck

Fast as hell (Lookup time is 🔥 compared to lists)

Organized AF (No random indexes to remember)

Can store anything (Numbers, text, lists, even other dictionaries)

How to Add, Remove, and Mess with Data

If you wanna add a new item, just do:

```
gang_dict["power"] = "Infinite Roasts"
```

Boom! You just added a new key-value pair.

If you wanna remove some sh*t, do this:

```
del gang_dict["age"]
```

Poof! It’s gone.

If you wanna get a value safely without Python bitching about missing keys, use:

```
gang_dict.get("name", "Default Value")
```

If you wanna see all the keys like a f*cking hacker, use:

```
gang_dict.keys()
```

And if you wanna flex with all the values, do:

```
gang_dict.values()
```

To see the whole ass dictionary, use:

```
gang_dict.items()
```

When to Use Dictionaries?

If you want f*cking quick lookups (Instead of scrolling through lists like a loser)

If you need structured data (Names, addresses, stats, etc.)

If you don’t want to remember dumb index numbers

💀💀 Aight, here’s your homework, you lazy-ass coder:

1. Build Your Own Gangster Dictionary

Make a dictionary with at least 5 key-value pairs describing your dream gangster character (e.g., name, power, weapon, weakness, catchphrase). Print each value separately.

2. Dictionary Operations Like a Boss

Add a new key called "archenemy" and assign it a name.

Delete the "weakness" key.

Print all the keys.

Print all the values.

Print the entire dictionary.

3. Create a Nested Dictionary (Gang War Edition)

Make a dictionary where each key is a gangster’s name, and the value is another dictionary containing their power level, weapon, and rank. Print everything in a clean format.

4. Dictionary Speedrun Challenge

Write a script where you:

Ask the user to input a word

Check if the word exists in your slang dictionary (you have to make one with at least 10 words)

If it exists, print the meaning. If not, print "Bro, tf is that? 💀"

🔷 Bonus: Let the user add a new word if it doesn’t exist.

👀 Try this sh*t, or else Python will haunt you in your dreams.

💀 Lists are like broke guys living on rent. Dictionaries are f*cking landlords.

Chapter X: Stop Being a Weak-Ass Noob and Learn to Code, Bitch

So you thought learning Python was just about reading some soft-ass tutorial and calling yourself a coder? Nah, bitch, this ain't a fcking bedtime story. You gotta put in the WORK. If you're still out here copy-pasting code like a damn parrot, close this book and go back to watching 🟧⬛, 'cause real coders don't have time for weak-ass fools like you.

Why Are You So Dumb?

Let me guess—
🔷 You "tried" coding for 2 minutes and rage quit?
🔷 You Googled "how to hack NASA" instead of learning basic-ass syntax?
🔷 You wrote one print("Hello, World!") and thought you were a fcking software engineer?

💀💀💀 BITCH, WAKE THE FCK UP.

If you don't understand loops, conditions, or functions by now, you got TWO options:

1. Stop crying, open your editor, and start fcking CODING.

2. Quit like a weak-ass loser and go back to scrolling reels.

Real Coders Don't Bitch About Errors

Errors are your best friend. If you're crying about a SyntaxError, slap yourself and go fix it.
🔷 Real coders debug.
🔷 Real coders READ the error message.
🔷 Real coders don't ask dumb questions like "wHy mY cOdE nOt wOrK" without even checking the fcking code.

Stop acting like an NPC and start THINKING. 💀

Homework (If You're Not a Weak Bitch)

1. Write a Python script that prints numbers from 1 to 100 (without Googling).

2. Create a function that takes a number and checks if it's even or odd.

3. Don't ask dumb-ass questions before actually TRYING the code.

If you can't do these, congratulations, you just proved that your brain is running on 1 FPS. 💀 Go back, re-read, and start again.

If you actually made it this far, you might have a chance of not being a fcking failure. Keep going, or quit and work at McDonald's. Your choice.

Chapter 11: File Handling – Because Not Everything Belongs in RAM, Dumbass

Aight, now that you know how to store data in lists and dictionaries, let's talk about storing shit permanently. Because let's be real—your memory is worse than a goldfish, and your code's memory resets every time you run it.

That’s where file handling comes in. Python lets you read, write, and update files like a pro, so your data doesn’t get lost like your will to live after debugging for 6 hours.

Reading Files (a.k.a. Stealing Data Like a Pro)

Let’s say you have a file called dumbass.txt that contains this:

Bro, stop watching 🟧⬛ and start coding.

You can read it like this:

```
file = open("dumbass.txt", "r")  # "r" means read mode
content = file.read()
file.close()  # Always close the file, dumbass
print(content)
```

Output:

```
Bro, stop watching 🟧⬛ and start coding.
```

💀 Congrats, you just learned how to read like a 5-year-old.

Writing Files (a.k.a. Leaving Your Dumb Thoughts on Paper)

Wanna write some shit into a file?

```
file = open("dumbass.txt", "w")  # "w" means write mode
file.write("If you're still struggling, maybe coding isn’t for you.")
file.close()
```

This overwrites everything in dumbass.txt.
💀 So if you had your mom’s Netflix password in there, it's gone.

Appending (Adding More Dumb Thoughts)

If you wanna add stuff to a file instead of erasing it like a dumbass, use "a" (append mode).

```
file = open("dumbass.txt", "a")  # "a" means append
file.write("\nNah, just kidding. Keep coding.")
file.close()
```

This adds the new text instead of deleting the old one.

Reading Files Like a Smartass

Instead of reading everything at once, you can read line by line like a civilized human.

```
file = open("dumbass.txt", "r")
for line in file:
    print(line.strip())  # strip() removes extra newlines
file.close()
```

💀 Now you’re officially better than 90% of Python beginners.

Using with – Because You’re Too Dumb to Close Files

Forgetting to close() a file is like leaving your 🟧⬛ tab open when your mom walks in. Disaster.
So Python gives us a smarter way:

```
with open("dumbass.txt", "r") as file:
    content = file.read()
print(content)  # File closes automatically, idiot
```

💀 No more worrying about closing files like a caveman.

Summary – What You Just Learned (If You Weren't Daydreaming Like an NPC)

✅ "r" = Read
✅ "w" = Write (overwrites everything)
✅ "a" = Append (adds new stuff)
✅ Use with open(...) so you don't forget to close files

💀 You just unlocked a new skill. Now stop crying about losing data and start storing it properly.

Chapter 12: Exception Handling – Because Your Code Will Always Fcking Break

Alright, listen up, dumbass. No matter how good you think you are at coding, your program WILL crash. And when it does, you're gonna sit there crying like a baby unless you learn how to handle exceptions properly.

An exception is basically Python saying:

> "Bro, your code is dumb, and I have no clue what to do now."

Instead of letting your program crash like a budget airline, you can catch errors and handle them like a pro.

Basic Try-Except (A.K.A. "Oh Shit, Catch That Error")

Let's start with an example of what happens when you're an idiot:

```
num = int(input("Enter a number: "))  # User enters 'hi'
print(10 / num)
```

If you type anything other than a number, Python throws a ValueError and your program dies instantly.

Fix it like a smartass:

```
try:
    num = int(input("Enter a number: "))
    print(10 / num)
except ValueError:
    print("Bro, you had one job. Enter a NUMBER.")
except ZeroDivisionError:
    print("Are you dumb? You can't divide by zero.")
```

💀 Now your code won't die if the user is a dumbass.

Catching Every Error (A.K.A. "I'm Too Lazy to Debug")

If you don't know which error will happen (or you're just lazy):

```
try:
    num = int(input("Enter a number: "))
    print(10 / num)
except Exception as e:
    print(f"Congrats, you broke it: {e}")
```

💀 Now it'll tell you exactly what went wrong.

Finally Block (A.K.A. "Clean Up Your Mess, Idiot")

What if your code opens a file, connects to a server, or steals FBI data and then crashes?
💀 You gotta clean that shit up.

```
try:
    file = open("important.txt", "r")
    print(file.read())
except FileNotFoundError:
    print("Bruh, that file doesn't even exist.")
finally:
    print("Whatever happened, at least this runs.")
```

💀 The finally block ALWAYS runs. Even if your PC explodes.

Raise Your Own Errors (A.K.A. "You Fcked Up, Now Suffer")

Python lets you throw your own errors like a savage:

```
age = int(input("Enter your age: "))

if age < 18:
    raise ValueError("Bro, you're too young for this shit.")
else:
    print("Welcome, legend.")
```

💀 Now your program will literally refuse to run if conditions aren't met.

Summary – What You Just Learned (If You Weren't Sleeping, NPC)

✅ try – Run your code normally
✅ except – Catch errors like a pro
✅ finally – Runs no matter what
✅ raise – Throw errors like a savage

💀 Congrats, now your code won't crash like a Windows XP.
Now go fix those dumb errors you've been ignoring.

Chapter 13: Object-Oriented Programming (OOP) – The Only Way to Act Like a Pro

Alright, dumbass, it's time to level up. If you've been coding like a caveman using only functions and variables, you're about to become a fcking wizard. Welcome to Object-Oriented Programming (OOP), the reason why real programmers flex on noobs like you.

WTF is OOP?

OOP is just a fancy way of organizing code using classes and objects.

💀 Imagine you're a game dev making a GTA clone.
You don't wanna write separate code for every single NPC, car, or stripper, right?
Instead, you create a class (like a blueprint), and then you can spawn as many objects as you want.

Example: If “Car” is a class, then “Lamborghini, Truck, and Rickshaw” are objects.

Creating a Class (A.K.A. "Making Your Own NPC")

Here’s how you define a class:

```
class Car:
    def __init__(self, brand, speed):
        self.brand = brand
        self.speed = speed

    def drive(self):
        print(f"{self.brand} is driving at {self.speed} km/h!")
```

💀 __init__ is like the birth certificate. It gives properties to the object.

Creating an Object (A.K.A. "Spawning Your Own Car")

Now that we have a class, let’s create some actual cars:

```
lambo = Car("Lamborghini", 320)
rickshaw = Car("Rickshaw", 50)

lambo.drive()  # Output: Lamborghini is driving at 320 km/h!
rickshaw.drive()  # Output: Rickshaw is driving at 50 km/h!
```

💀 BOOM! We just made two different cars using the same class.

Adding More Functions (A.K.A. "Making Shit Happen")

You can add methods (a.k.a. functions inside a class) to make objects do more stuff:

```
class Car:
    def __init__(self, brand, speed):
        self.brand = brand
        self.speed = speed

    def drive(self):
        print(f"{self.brand} is driving at {self.speed} km/h!")

    def nitro(self):
        self.speed += 50
        print(f"{self.brand} used Nitro! New speed: {self.speed} km/h!")
```

Now try this:

```
lambo = Car("Lamborghini", 320)
lambo.nitro()  # Output: Lamborghini used Nitro! New speed: 370 km/h!
```

💀 Now our Lambo got a speed boost.

Inheritance (A.K.A. "Why Copy-Paste When You Can Be Smart?")

If you have common shit between classes, don’t repeat yourself.
Just make a base class and let others inherit from it.

```
class Vehicle:
    def __init__(self, brand, speed):
        self.brand = brand
        self.speed = speed
    def drive(self):
        print(f"{self.brand} is driving at {self.speed} km/h!")

class Bike(Vehicle):
    def wheelie(self):
        print(f"{self.brand} is doing a wheelie at {self.speed} km/h!")
```

Now, let’s spawn some bikes:

```
ninja = Bike("Kawasaki Ninja", 200)
ninja.drive()  # Output: Kawasaki Ninja is driving at 200 km/h!
ninja.wheelie()  # Output: Kawasaki Ninja is doing a wheelie at 200 km/h!
```

💀 No need to rewrite drive(), we inherited it from Vehicle!

Summary – What You Just Learned (If Your Brain Didn’t Melt)

✅ class – Creates a blueprint
✅ __init__ – Gives objects their properties
✅ self – Refers to the object itself
✅ Methods – Functions inside a class
✅ Objects – Actual instances of a class
✅ Inheritance – Avoids copy-pasting shit

💀 Congrats, now you can code like a fcking pro.
Now go make your own NPCs, cars, and weapons.

Chapter 14: Debugging – Fix Your Shitty Code, You Caveman

Alright, you piece of monkey brain, if you’ve made it this far, I guarantee you’ve written some absolute dogshit code. And guess what? It doesn’t fcking work.

You’re sitting there, staring at the screen like an idiot, thinking, “Bruh why tf my code ain’t working?”
BECAUSE YOU WROTE GARBAGE, THAT’S WHY.

Now, instead of crying like a little b*tch, let me teach you how to debug and fix your disasters like a real coder.

Step 1: Read the Error Message, Dumbass

When Python throws an error, it literally tells you what’s wrong.

💀 But nah, your dumbass be like:

> “HELP MY CODE IS BROKEN 😭😭”

READ THE DAMN ERROR MESSAGE.

Example:

```
print(5 / 0)
```

🔥 Error:

```
ZeroDivisionError: division by zero
```

🚨 Bitch, you just tried to divide by zero. That shit is illegal in math and in life.

Step 2: Use Print Statements Like a Detective

When your code messes up, don't just sit there like a potato.

Add print statements and check WTF is happening.

Example:

```
def add_numbers(a, b):
    print("A:", a, "B:", b)  # Checking inputs
    return a + b

result = add_numbers(5, "10")  # ERROR
print("Result:", result)
```

🔥 Error:

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

💀 Congrats, dumbass, you tried to add a number and a string.

Step 3: Use a Damn Debugger (If You're Not a Caveman)

If printing everything makes you feel like a caveman, use a real debugger.

In VS Code, PyCharm, or any IDE, you can:
✅ Set breakpoints (pause code at a certain line)
✅ Step through code (run one line at a time)
✅ See variable values (no more print spam)

🚨 But nah, your dumbass will still print("WTF") and try to guess.

Step 4: Google is Your Best Friend, Stop Acting Smart

You think real programmers just know everything?

HELL NO.

They Google errors like crazy.

💀 Meanwhile, you be sitting there like:

> "I'mma fix this myself."

BRO, JUST GOOGLE IT.
99% of the time, some other dumbass already asked the same question on Stack Overflow.

Step 5: Ask for Help (But Don't Be a Clown About It)

If you're still lost, ask for help – but don't be an idiot.

🚨 What NOT to do:
❌ "My code is broken. Help."
❌ "It's not working. Fix it."
❌ "Error. Pls solve."

🚀 What TO do:
✅ Explain what's wrong
✅ Show your error message
✅ Post your code snippet

💀 Otherwise, no one is helping your lazy ass.

Step 6: Accept That You're a Dumbass and Move On

Debugging takes practice. You'll keep making stupid mistakes, and that's fine.

But if you don't fix them, you're just a clown. 🤡

🔥 Follow these steps:
✅ READ the error message
✅ PRINT statements to check values
✅ USE a debugger like a smart person
✅ GOOGLE like a pro
✅ ASK for help properly

Now stop crying and fix your shitty code. 💀

Chapter 15: Spanking Your Dumbass into a Real Coder ✋🍑

Alright, listen up, you lazy sack of pixels. You've been goofing around, copy-pasting code, and thinking you're a badass programmer—but in reality, your code is ass.

This chapter is your wake-up call. Time to stop being a wannabe script kiddie and actually code like a real chad.

1. Stop Copy-Pasting Like a Dumbass

Let me guess, you:

Copy-paste code from Stack Overflow

Don't even read it

Wonder why it doesn't work

💀 Bruh, you didn't write that code. It doesn't know you.

🔥 REALITY CHECK:
If you copy-paste without understanding, you're just a dumbass data entry operator, not a programmer.

🚀 Fix yourself:
✅ Read the code
✅ Understand every line
✅ Modify it to see what changes

2. Learn to Type Fast or Quit Programming

You out here typing like a paralyzed snail? Nah, bro, that's not it.

🔥 Real programmers type like gods.
If you type slower than a grandma texting, you'll NEVER be a real coder.

💀 Imagine debugging a 500-line code at 10 WPM. You'll die before you fix the bug.

🚀 Fix yourself:
✅ Practice typing every day
✅ Use keybr.com or TypeRacer
✅ Aim for at least 60 WPM

3. Stop Acting Smart If You Don't Know Shit

Some of y'all be out here acting like tech gods, but your code doesn't even run.

Example of a dumbass conversation:
👤 You: "Bro, I made an AI chatbot!"
💀 Reality: It's just an if-else script.

👤 You: "I made an advanced algorithm!"
💀 Reality: You copied Bubble Sort from Wikipedia.

🔥 REALITY CHECK:
Nobody cares about your fake flex. Just admit you don't know and learn properly.

🚀 Fix yourself:
✅ Shut up and code
✅ Learn new concepts DAILY
✅ Actually build projects instead of talking

4. If You Give Up Easily, You're Weak AF

🔥 Coding is PAIN. 🔥
There is no easy way to learn it.

If you rage-quit because of:
❌ A missing semicolon
❌ An error you don't understand
❌ Bugs that won't go away

Then pack your bags and leave. Programming is not for the weak.

🚀 Fix yourself:
✅ Debug for HOURS if needed
✅ Take breaks, but NEVER quit
✅ Become a bug-hunting demon

5. The Only Way to Git Gud is to Code Daily

If you think watching tutorials makes you a programmer, you're dumber than a bag of rocks.

🔥 Real programmers code every day.
Watching videos = Theory
Writing code = Skill

🚀 Fix yourself:
✅ Code EVERY DAY
✅ Make small projects
✅ Push to GitHub (if you know how)

💀 If you code once a week, your brain is deleting everything.

6. Accept That Your First 100 Projects Will Be Dogshit

You're not Elon Musk after writing your first Python script. Your code will be:
💀 Ugly
💀 Slow
💀 Messy

🔥 REALITY CHECK:
EVERYONE starts as a shitty programmer.
The difference? Some keep going. Some quit.

🚀 Fix yourself:
✅ Keep writing shit code
✅ Improve a little each time
✅ Look back after a year and laugh at your old dumbass

7. No One is Coming to Save You. Learn On Your Own.

You keep waiting for someone to teach you? Wake up.
🔥 Self-learning is the only way to win.

💀 No one is holding your hand.
If you keep asking dumbass questions like "Where do I start?", you'll NEVER start.

🚀 Fix yourself:
✅ Google everything
✅ Read documentation (stop being scared)
✅ Solve your own problems

Final Words: Grow Some Balls & Code Like a Beast

You either:
🔥 Grow some balls and learn programming properly
💀 Stay a loser who just copies shit

Your choice.

Now get your ass up and write some REAL code. ✋🍑

Chapter 16: Homework, Because Your Lazy Ass Needs Practice 🤡

Alright, you survived the roast session in the last chapter (hopefully). Now it's time to actually code instead of just laughing at curse words.

I know you're lazy as hell, so I'm giving you small but deadly assignments. Do them, or stay a useless script kiddie forever.

1. The Basic Bitch Challenges (For Your Weak Brain 😘)

1.1 Hello, World? Nah, Hello, Dumbass!

Write a Python program that asks for your name and then insults you with a random roast.

Example:

```
Enter your name: Rahul
Output: Rahul, you type slower than a Nokia 3310 keyboard. Get better.
```

🔷 Hint: Use input() and random.choice().

1.2 The “Can’t Even Do Math” Challenge

Write a calculator program that takes *two numbers and an operator (+, -, , /) and prints the result.

❌ If they enter something dumb (like dividing by zero), insult them.

Example:

Enter first number: 5
Enter operator: /
Enter second number: 0
Output: Are you dumb? You can’t divide by zero.

🔷 Hint: Use if-else, and check for division by zero.

2. The Slightly Less Dumb Challenges 🤔

2.1 The Password Strength Checker

Write a program that checks if a password is strong or weak.

💀 Weak password if:

Less than 8 characters

Doesn’t have a number

Doesn’t have a special character

🔥 Strong password if:

More than 8 characters

Has numbers, uppercase, lowercase, and symbols

Example:

Enter password: password123
Output: Weak as hell. Go fix your life.

🔷 Hint: Use re (regex) to check conditions.

2.2 The “Guess The Number or Stay a Loser” Game

Make a number guessing game.

The program randomly picks a number between 1-100

The user gets 5 chances to guess it

If they guess wrong, give a hint (higher/lower)

If they fail all 5 tries, roast them

🔷 Hint: Use random.randint() and a loop.

3. The “You’re Not Ready But Do It Anyway” Challenges 💀

3.1 The File Organizer

Write a Python script that:

Scans a folder

Moves images, videos, and documents to separate folders

Example:

```
Folder before:
  - image.jpg
  - video.mp4
  - document.pdf
  - random.exe

Folder after:
  /Images  -> image.jpg
  /Videos  -> video.mp4
  /Docs    -> document.pdf
```

🔷 Hint: Use os and shutil.

3.2 The Auto Spammer (For Educational Purposes 💀)

Make a program that takes a message and a number, then spams that message in a chat box.

⚠️ WARNING:
DO NOT use this for illegal shit. This is just for fun in a text file or notepad.

🔷 Hint: Use pyautogui.

SUBMISSION INSTRUCTIONS (DO IT OR YOU'RE A FAILURE 💀)

🔥 Complete at least 3 assignments.
🔥 Send your Python files to dwipbiswas@yahoo.com.
🔥 Subject: "I Got My Ass Roasted, Now Here's My Code"
🔥 If you don't submit, you're officially a certified script kiddie.

CERTIFICATE OF DUMBASSARY 🎖️

If you successfully complete the assignments without errors, you will receive a prestigious digital "Certificate of Dumbassary", proving that you:
✅ Survived this book
✅ Actually wrote Python code instead of watching 🟧⬛
✅ Graduated from being a brain-dead beginner

💀 Now stop reading, start coding, and EARN your respect.

Chapter 17: Type Conversion in Python – Stop Mixing Shit Like an Idiot 💀

Alright, listen up. If you're still mixing strings with numbers and wondering why your code throws errors like a drunk uncle, this chapter is for you.

1. What Is Type Conversion?

In Python, data types don't always play nice with each other. You can't mix an integer with a string like it's some cocktail.

💀 Example of a dumbass mistake:

```
age = 15
print("Your age is " + age)  # ❌ ERROR! Can't add int and str
```

🔥 Fix it using type conversion:

```
age = 15
```

```
print("Your age is " + str(age))  # ✅ Now it works
```

2. Types of Type Conversion

2.1 Implicit Conversion (Python Fixes Your Dumb Mistakes)

Python automatically converts one data type to another if it makes sense.

✅ Example (works fine, no errors):

```
num = 5   # int
decimal = 2.5  # float
result = num + decimal  # int + float → float
print(result)  # Output: 7.5
```

Python silently converts num from int to float. It's like magic, but not really.

2.2 Explicit Conversion (You Fix Your Own Mistakes)

Sometimes Python doesn't convert things automatically, so you have to force it.

🔷 Common functions:

int() → Converts to an integer

float() → Converts to a float

str() → Converts to a string

list() → Converts to a list

tuple() → Converts to a tuple

dict() → Converts to a dictionary

🔥 Example:

```
x = "10"  # It's a string
y = int(x)  # Now it's an integer
print(y + 5)  # Output: 15
```

💀 If you don't convert, you'll get errors.

❌ WRONG:

```
x = "10"
print(x + 5)  # TypeError: Can't add str and int
```

✅ RIGHT:

```
x = "10"
print(int(x) + 5)  # Output: 15
```

3. Common Type Conversion Problems (Aka Fix Your Shit 💀)

3.1 Converting User Input (Because Input Is Always a String)

When you take input from a user using input(), it's always a string.

❌ WRONG:

```
age = input("Enter your age: ")
```

```
print(age + 5)  # TypeError
```

✅ RIGHT:

```
age = int(input("Enter your age: "))  # Convert to int
print(age + 5)  # Works fine
```

3.2 Converting Lists and Tuples

Sometimes you need to convert between lists, tuples, and sets.

🔥 Example:

```
my_list = [1, 2, 3]
print(tuple(my_list))  # Converts list to tuple → (1, 2, 3)
print(set(my_list))  # Converts list to set → {1, 2, 3}
```

❌ But converting a dictionary is different.

🔥 Example:

```
my_dict = {"a": 1, "b": 2, "c": 3}
print(list(my_dict))  # Output: ['a', 'b', 'c'] (Only keys get converted)
```

Chapter 18: Python Libraries – Stop Reinventing the Wheel, Dumbass 💀

Bro, Python is famous because of its libraries.
If you're not using libraries, you're just wasting time.

1. What Is a Library?

A library is a collection of pre-written code that makes your life 100x easier.

Example:

Want to do math? Use NumPy.

Want to manipulate data? Use Pandas.

Want to build a website? Use Flask or Django.

Want to automate boring shit? Use Selenium.

💀 If you don't use libraries, you're basically coding with a stone and a stick.

2. The Most Popular Python Libraries

2.1 NumPy – Because Python Sucks at Math

If you try doing advanced math in Python without NumPy, you'll want to throw your PC out the window.

🔥 Example (Matrix Addition using NumPy):

```
import numpy as np

a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

print(a + b)
```

2.2 Pandas – The Data Science Lifeline

If you work with CSV files or large datasets, Pandas will save your life.

🔥 Example (Reading a CSV file):

```
import pandas as pd

data = pd.read_csv("data.csv")  # Reads the file
print(data.head())  # Shows first 5 rows
```

2.3 Matplotlib – For Making Graphs and Charts

If you need to visualize data, use Matplotlib.

🔥 Example (Basic Line Graph):

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.plot(x, y)
plt.show()
```

2.4 Requests – Web Scraping for the Lazy

If you need to download data from websites, use requests.

🔥 Example (Getting Data from a Website):

```
import requests

response = requests.get("https://jsonplaceholder.typicode.com/posts/1")
print(response.json())  # Shows JSON data from the website
```

2.5 Flask – Build Websites Like a Pro

🔥 Example (Simple Web Server):

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Welcome to my website!"

app.run()
```

3. Installing Libraries (Don't Just Copy-Paste and Cry 💀)

Most libraries are not built-in. You need to install them first.

🔷 To install a library, use:

```
pip install numpy
pip install pandas
pip install flask
```

💀 If you don’t install them, your code will cry with an ImportError.

Final Words

Stop doing everything manually.
Use libraries. Save time. Look smart.

💀 If you're still manually writing math functions, you probably enjoy pain.

🔥 Next up: More advanced Python concepts. Get ready.

Chapter 19: Accumulators – Stop Being a Dumbass and Start Counting Properly 💀

Alright, if you’ve ever tried adding numbers in a loop but somehow ended up with a TypeError or a random ass bug, you need to learn about accumulators.

💀 Accumulators = Variables that keep track of a running total.
🔥 Think of it as a bucket where you keep adding stuff until you’re done.

1. What Is an Accumulator?

An accumulator is a variable that stores a running total while looping through data.

💀 If you don’t use an accumulator, you’ll be stuck doing manual math like a caveman.

🔥 Basic Example:

```
total = 0  # Start with zero

for i in range(1, 6):  # Loop from 1 to 5
    total += i  # Add i to total

print(total)  # Output: 15 (1+2+3+4+5)
```

💀 If you forgot to initialize total = 0, Python would have roasted your ass with an error.

2. Types of Accumulators

2.1 Numeric Accumulators (For Counting or Summing)

🔥 Example 1: Summing Numbers

```
sum = 0
for num in [3, 5, 7, 9]:
    sum += num  # Add each number to sum

print(sum)  # Output: 24
```

🔥 Example 2: Counting Items

```
count = 0
for char in "hello":
    count += 1  # Count each character

print(count)  # Output: 5
```

2.2 String Accumulators (For Joining Text)

If you concatenate strings inside a loop, use a string accumulator instead of printing every time (which is dumb and inefficient).

🔥 Example: Joining a Sentence

```
sentence = ""
words = ["Python", "is", "cool"]

for word in words:
    sentence += word + " "  # Add word + space

print(sentence.strip())  # Output: "Python is cool"
```

💀 If you used print() inside the loop instead, your output would look dumb and split.

2.3 List Accumulators (For Collecting Data)

🔥 Example: Storing Even Numbers in a List

```
evens = []  # Start with an empty list

for num in range(1, 11):  # Loop from 1 to 10
    if num % 2 == 0:
        evens.append(num)  # Add even numbers

print(evens)  # Output: [2, 4, 6, 8, 10]
```

💀 If you didn't use .append(), you'd be crying over an empty list.

2.4 Dictionary Accumulators (For Counting Occurrences)

🔥 Example: Counting Letter Frequency in a String

```
freq = {}

for char in "banana":
    freq[char] = freq.get(char, 0) + 1  # Increase count

print(freq)  # Output: {'b': 1, 'a': 3, 'n': 2}
```

💀 If you didn't use .get(), Python would throw a KeyError like an asshole.

3. Common Accumulator Mistakes (Fix Your Shit 💀)

❌ Forgetting to Initialize the Accumulator

```
for i in range(5):
    total += i  # ❌ ERROR! total is not defined
```

🔥 ✅ Correct Way:

```
total = 0
for i in range(5):
    total += i
```

❌ Using print() Inside the Loop Instead of an Accumulator

```
for i in range(1, 6):
    print(i + i)  # ❌ Prints every step separately (dumb)
```

🔥 ✅ Correct Way (Using an Accumulator):

```
total = 0
for i in range(1, 6):
    total += i + i  # Accumulate instead of printing each step
print(total)  # ✅ Correct total
```

4. When to Use Accumulators?

🔥 Use accumulators when:
✅ You need to sum values in a loop.
✅ You need to count occurrences of something.
✅ You need to store collected values in a list.
✅ You need to build a string dynamically.

Final Words

💀 If you don't use accumulators, your loops are probably useless.
🔥 Start tracking values properly instead of writing random prints and crying.

📢 Next up: More Python roasting & pro-level coding. Get ready.

Chapter 20: Welcome, Morons – Menu-Driven Programs 💀

Alright, you lazy ass coders, it's time to stop running your scripts one by one like a damn amateur.
🔥 Today, we're making a MENU-DRIVEN PROGRAM – a script that lets users choose options like a boss instead of running individual scripts like a caveman.

💀 If you don't know what a menu-driven program is, congratulations, you've been coding like a dumbass this whole time.

1. What the Hell is a Menu-Driven Program?

A menu-driven program lets users pick options from a menu instead of manually running different parts of a script.

💀 If your program makes users type random inputs without giving them choices, your UI is trash.

🔥 Example of a Menu-Driven Program:

```
Welcome, Morons!
1. Add Numbers
2. Subtract Numbers
3. Multiply Numbers
4. Exit
Enter your choice: _
```

💀 User types 1? Boom. Addition function runs.
💀 User types 4? Get lost, program exits.

2. The Basic Structure of a Menu-Driven Program

🔥 Basic Example:

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b
```

```
def multiply(a, b):
    return a * b

while True:  # Infinite loop to keep the menu running
    print("\nWelcome, Morons! 💀")
    print("1. Add Numbers")
    print("2. Subtract Numbers")
    print("3. Multiply Numbers")
    print("4. Exit")

    choice = input("Enter your choice: ")

    if choice == '1':
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))
        print("Result:", add(a, b))

    elif choice == '2':
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))
        print("Result:", subtract(a, b))

    elif choice == '3':
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))
        print("Result:", multiply(a, b))

    elif choice == '4':
        print("Alright, get lost. 💀")
        break  # Exit the loop

    else:
        print("Bruh, enter a valid option. 💀")
```

3. How This Works?

1. It loops forever (until the user chooses exit).

2. It prints a menu so the dumb user knows what to do.

3. It takes input and runs the corresponding function.

4. It checks for invalid inputs and roasts the user if needed.

🔥 Run this, and you'll get a working menu system like an actual coder.

4. Improving This Dumbass Code

🔥 Adding a Clear Screen Function (For Better UI)

```
import os

def clear_screen():
    os.system('cls' if os.name == 'nt' else 'clear')  # Clears the screen

while True:
    clear_screen()  # Clears screen before showing menu
    print("Welcome, Morons! 💀")
    print("1. Add Numbers")
    print("2. Subtract Numbers")
    print("3. Multiply Numbers")
    print("4. Exit")

    choice = input("Enter your choice: ")

    if choice in ['1', '2', '3']:
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))

    if choice == '1':
        print("Result:", add(a, b))
    elif choice == '2':
        print("Result:", subtract(a, b))
    elif choice == '3':
        print("Result:", multiply(a, b))
    elif choice == '4':
        print("Alright, get lost. 💀")
        break
    else:
        print("Bruh, enter a valid option. 💀")

    input("\nPress Enter to continue...")  # Pause before showing menu again
```

💀 Now the menu refreshes every time instead of cluttering the terminal.

5. Expanding The Menu – More Features

🔥 Let's add more dumbass features to flex our skills.

Adding a Calculator with More Functions

```
import math

def divide(a, b):
    return a / b if b != 0 else "Bruh, don't divide by zero 💀"

def square_root(a):
    return math.sqrt(a)

while True:
    clear_screen()
    print("Welcome, Morons! 💀")
    print("1. Add Numbers")
    print("2. Subtract Numbers")
    print("3. Multiply Numbers")
    print("4. Divide Numbers")
    print("5. Square Root")
    print("6. Exit")

    choice = input("Enter your choice: ")

    if choice in ['1', '2', '3', '4']:
        a = float(input("Enter first number: "))
        b = float(input("Enter second number: "))

    if choice == '1':
        print("Result:", add(a, b))
    elif choice == '2':
        print("Result:", subtract(a, b))
    elif choice == '3':
        print("Result:", multiply(a, b))
    elif choice == '4':
        print("Result:", divide(a, b))
    elif choice == '5':
        a = float(input("Enter a number: "))
        print("Square Root:", square_root(a))
    elif choice == '6':
        print("Alright, get lost. 💀")
        break
    else:
        print("Bruh, enter a valid option. 💀")
    input("\nPress Enter to continue...")
```

🔥 Now you got a proper calculator menu that can do basic math.

6. Why The Hell Do You Need This?

💀 If you're still manually running different scripts, you're a clown.
💀 Menus make your program look professional instead of a random mess.
💀 You can keep adding features without rewriting your whole code.

7. Next Up: File Handling & Making a Real CLI App

Now that you got a basic menu, let's level up and make a proper CLI app that saves data.
💀 Stay tuned, dumbasses, we're getting pro soon.

Chapter 21: Parameters – Stop Writing Dumbass Code 💀

Alright, you absolute buffoons, today we're talking about parameters.
🔥 If you're writing functions without parameters, you're doing it WRONG.
🔥 If you're copying the same code multiple times instead of using parameters, you're a damn caveman.

1. What The Hell Are Parameters?

💀 A parameter is a variable that gets passed into a function.
💀 It allows a function to take input and actually be useful instead of a dumb fixed script.

🔥 Example of a Dumbass Function Without Parameters:

```
def greet():
    print("Hello, you dumb moron!")
greet()
```

💀 This function is TRASH because it always prints the same thing.

🔥 Now, let's add a parameter and make it actually useful:

```
def greet(name):
    print(f"Hello, {name}! Stop being dumb.")
greet("Dwip")
greet("Random Clown")
```

💀 Now, it can greet anyone instead of being useless.

2. Types of Parameters in Python

Python has 4 types of parameters, and if you don't learn them, I swear you deserve to be slapped.

1️⃣ Positional Parameters (Normal Parameters)

These are the basic parameters you pass into a function in order.
🔥 Example:

```
def roast(name, insult):
    print(f"{name}, you are a {insult} 💀")

roast("Dwip", "dumbass")
roast("Random Kid", "clown")
```

💀 If you mess up the order, you mess up the roast.

2️⃣ Default Parameters

These are parameters with a default value. If you don't pass a value, Python won't cry like a baby.
🔥 Example:

```
def roast(name, insult="idiot"):
    print(f"{name}, you are a {insult} 💀")

roast("Dwip")  # Uses the default insult
roast("Random Kid", "bozo")  # Uses custom insult
```

💀 Now, even if you forget the second argument, it still works.

3️⃣ Keyword Arguments (Named Parameters)

🔥 What if you forget the order? Just name the damn parameters.

```
def roast(name, insult):
    print(f"{name}, you are a {insult} 💀")

roast(insult="clown", name="Dwip")  # Order doesn't matter now
```

💀 Even if you're dumb, Python still understands what you're doing.

4️⃣ Arbitrary Arguments (Unlimited Parameters, AKA *args)

🔥 What if you want to pass a random number of arguments?
🔥 Just use *args, and Python will shut up and accept anything.

```
def roast(*names):
    for name in names:
        print(f"{name}, you are a dumbass 💀")

roast("Dwip", "Random Kid", "Some Clown")
```

💀 You can pass 1 name, 5 names, or 500 names, and it will still work.

3. Combining Different Parameters Like a Pro

🔥 Let's combine everything like a goddamn legend.

```
def insult(name, *insults, default_insult="clown"):
    if insults:
        print(f"{name}, you are a {', '.join(insults)} 💀")
    else:
        print(f"{name}, you are a {default_insult} 💀")

insult("Dwip", "dumbass", "bozo", "idiot")
insult("Random Kid")
```

💀 Now, you can pass any number of insults, and if you don't, it uses a default one.

4. Return Values – Because Printing is for Babies

🔥 If your function only prints stuff, you're a clown. Make it RETURN values.

```
def add(a, b):
    return a + b  # Returns the result instead of printing

result = add(5, 10)
print("Sum:", result)  # Now you can use the result anywhere
```

💀 Now your function actually DOES something instead of just printing like an idiot.

🔥 Another Example:

```
def roast(name):
    return f"{name}, you are a dumbass 💀"

roast_msg = roast("Dwip")
print(roast_msg)  # Now we can store it, send it, do whatever
```

💀 Stop printing everything inside functions, use return.

5. TL;DR – Quick Recap for Your Dumb Brain

💀 Parameters = Function Inputs.
💀 Positional Parameters → Pass values in order.
💀 Default Parameters → No input? Uses a default value.
💀 Keyword Arguments → Name the parameters to avoid messing up order.
💀 Arbitrary Arguments (*args) → Accepts unlimited inputs.
💀 Return Values → Instead of printing, return results so they can be used later.

🔥 Now go write proper functions and stop coding like a dumbass.

Chapter 22: Constructors – Stop Writing Dumbass Code Like a Caveman 💀

Alright, you dumb clowns, if you've been writing Python classes without a constructor,
I swear you need to be slapped with a wet slipper 💀.

🔥 A constructor is an automatic setup function that runs when you create an object.
🔥 It's what makes your class actually useful, instead of being a dead body.

1. What The Hell is a Constructor?

💀 A constructor is a special method inside a class that gets called automatically when you create an object.
💀 It is always named __init__(), because Python loves double underscores like a weirdo.

🔥 Example of a Dumbass Class Without a Constructor:

```
class DumbPerson:
    def greet(self):
        print("Hi, I am dumb.")

p = DumbPerson()
p.greet()
```

💀 This class is trash because it doesn't set any data when we create an object.

🔥 Now, let's add a constructor and make it actually useful:

```
class Person:
    def __init__(self, name, age):  # Constructor with parameters
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hi, I am {self.name}, and I am {self.age} years dumb.")

p = Person("Dwip", 15)  # Creating an object
p.greet()
```

💀 Now, the class actually stores data and doesn't act like an empty NPC.

2. What Does self Mean?

🔥 self is a must in class methods. If you forget it, Python will slap you with an error.
🔥 It represents the current object and allows you to store data inside it.

💀 Look at this example:

```
class Clown:
    def __init__(self, name):
        self.name = name  # `self.name` stores the value inside the object

    def introduce(self):
        print(f"Hello, I am {self.name}, the biggest clown in Python.")

c = Clown("Dwip")
c.introduce()
```

💀 Without self, this wouldn't work.

3. Default Values in Constructors

🔥 You can give default values so idiots don't break your code.

```
class Noob:
    def __init__(self, name="Unknown", skill="None"):
        self.name = name
        self.skill = skill

    def show_skill(self):
        print(f"{self.name} is skilled in {self.skill}. Or maybe not.")

n1 = Noob()  # Uses default values
n2 = Noob("Dwip", "Roasting")

n1.show_skill()
n2.show_skill()
```

💀 Now even if someone forgets to pass values, it still works.

4. Multiple Constructors? Nah, Use Default Arguments!

🔥 Python doesn't support multiple constructors like Java. But we can fake it.

```
class Gamer:
    def __init__(self, name, game="Minecraft"):
        self.name = name
        self.game = game

    def play(self):
        print(f"{self.name} is playing {self.game}.")

g1 = Gamer("Dwip")  # Default game is Minecraft
g2 = Gamer("Random Kid", "Fortnite")  # Custom game

g1.play()
g2.play()
```

💀 Default values = Multiple constructor behavior.

5. TL;DR – Quick Recap for Your Dumb Brain

💀 A constructor (__init__()) is an automatic function that runs when an object is created.
💀 self stores data inside the object. Without self, you're dumb.
💀 Default values prevent your program from breaking when someone forgets an argument.
💀 Python doesn't allow multiple constructors, but default arguments work as a replacement.

🔥 Now stop writing empty classes and start using constructors properly.

Chapter 23: Destructors – When Your Code Dies Like Your Brain Cells 💀

Alright, listen up, you clowns. If you create objects, you better know how to destroy them too.
Otherwise, your RAM will suffer like your grades. 💀

🔥 A destructor is a method that runs automatically when an object is destroyed.
🔥 It helps free up memory, close files, or clean up resources.

1. What The Hell is a Destructor?

💀 Python automatically cleans up objects when they are no longer needed, BUT…
💀 Sometimes, you need to manually clean up stuff like database connections, files, or caches.
💀 That's where destructors (__del__()) come in.

🔥 Example of a Destructor in Action:

```
class Person:
    def __init__(self, name):
        self.name = name
        print(f"{self.name} was born.")

    def __del__(self):
        print(f"{self.name} just died. RIP 💀")

p1 = Person("Dwip")  # Creating an object
del p1  # Manually destroying the object
```

💀 When you run this, you'll see:

```
Dwip was born.
Dwip just died. RIP 💀
```

🔥 That's the destructor (__del__()) running when the object is deleted.

2. When Does Python Call the Destructor?

🔥 Python automatically calls the destructor when:
✅ The object goes out of scope (no longer needed).
✅ You manually delete it using del.
✅ Python garbage collector decides to remove it.

💀 Example of Automatic Destructor Call:

```
class NPC:
    def __init__(self, name):
        self.name = name
        print(f"{self.name} spawned.")

    def __del__(self):
        print(f"{self.name} despawned.")

def create_npc():
    n = NPC("Random Noob")  # This NPC exists only inside this function
    print("Doing some stuff with NPC...")

create_npc()
print("End of program.")
```

💀 Output:

```
Random Noob spawned.
Doing some stuff with NPC...
Random Noob despawned.
End of program.
```

🔥 Since n only exists inside the function, it gets destroyed automatically after the function ends.

3. Real-Life Use Cases – Not Just Some Theoretical BS

💀 1. Closing Files Properly:

```
class FileHandler:
    def __init__(self, filename):
        self.file = open(filename, "w")
        print("File opened.")

    def __del__(self):
        self.file.close()
        print("File closed.")

f = FileHandler("test.txt")
del f  # Ensures the file is closed
```

💀 2. Cleaning Up Memory:

```
class MemoryLeak:
    def __init__(self):
        print("Using a lot of memory...")

    def __del__(self):
        print("Cleaning up memory.")

obj = MemoryLeak()
del obj  # Clears the memory
```

💀 3. Disconnecting from a Database:

```
class Database:
    def __init__(self):
        print("Connected to database.")

    def __del__(self):
        print("Disconnected from database.")

db = Database()
del db
```

4. Can You Stop the Destructor from Running?

🔥 Yes, you can. If you're holding references to an object, Python won't destroy it.

```
class Ghost:
    def __init__(self, name):
        self.name = name
        print(f"{self.name} appeared.")

    def __del__(self):
        print(f"{self.name} vanished.")
```

```
g = Ghost("Casper")
ghost_ref = g  # Holding a reference
del g  # Destructor won't run yet

print("Object still exists!")
del ghost_ref  # Now the destructor will run
```

💀 Python won't delete the object until all references are gone.

5. TL;DR – Quick Recap for Your Dumbass Brain

💀 Destructors (__del__()) run when an object is destroyed.
💀 They help clean up memory, close files, or disconnect from databases.
💀 Python calls the destructor when the object is no longer needed OR when del is used.
💀 If an object still has references, it won't get destroyed.

🔥 Now you know how to kill objects. Stop leaving memory leaks like an idiot.

Chapter 24: Access Specifiers – Who TF Can Touch Your Code? 💀

Alright, dumbasses, listen up. You can't just let every fool access your variables and methods. That's like giving your phone password to your little cousin—they'll mess up everything.

🔥 That's where Access Specifiers come in.
They control who can access your class variables and methods.

1. Types of Access Specifiers in Python

Python has 3 types of access specifiers:

Public (variable_name) – Anyone can access it, like a slutty API 💀.

Protected (_variable_name) – Only inside the class & subclasses. (Like a VIP section 🤡)

Private (__variable_name) – Only inside the class. (Locked like your browser history 💀).

🔥 In Python, everything is public by default unless you explicitly protect it.

2. Public – Open to All Like a Free Wi-Fi 💀

💀 Public members can be accessed from anywhere.

```
class Person:
    def __init__(self, name):
        self.name = name  # Public Variable

    def greet(self):
        return f"Hello, my name is {self.name}"

p = Person("Dwip")
print(p.name)  # Works fine
print(p.greet())  # Works fine
```

Everything is accessible. Like a fool giving away their Netflix password.

3. Protected – Only for Family Members 💀

Protected variables start with a single underscore (_).

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self._age = age  # Protected Variable

    def show_age(self):
        return f"My age is {self._age}"

class Child(Person):
    def get_protected(self):
        return self._age  # Works fine inside subclass

p = Child("Dwip", 15)
print(p.show_age())  # Works fine
print(p.get_protected())  # Also works fine
print(p._age)  # Works but should be avoided
```

Accessing _age directly is possible but ugly.
It's like sneaking into VIP without permission.

4. Private – Keep That Shit Locked 🔒

Private variables start with double underscores (__).

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # Private Variable

    def show_balance(self):
        return f"Your balance is {self.__balance}"

b = BankAccount(10000)
print(b.show_balance())  # Works fine
print(b.__balance)  # ❌ Error! Can't access directly
```

💀 Trying to access __balance outside the class = ERROR.
It's like trying to hack into someone's bank account 💀.

5. Can You Hack a Private Variable?

💀 Python lets you access private variables using name mangling.

```
print(b._BankAccount__balance)  # Works! 💀
```

But if you do this, you're officially a dumbass.
Private variables exist for a reason—DON'T BREAK THE RULES.

Moral of the Story 💀

Public – Everyone can access. Like a public toilet 💀.

Protected – Only class & subclasses. Like a VIP party 🤡.

Private – Only inside the class. Like your hidden folder 💀.

Chapter 25: Inheritance – When Your Code Becomes Your Kid 💀

Alright, dumbasses, listen up. Inheritance is when a class gets traits from another class, just like you inherited stupidity from your ancestors.

💀 Think of it like this:

Parent Class (Super Class) = The OG, the boss.

Child Class (Sub Class) = The lazy ass who copies everything from the parent.

1. Why the F*ck Do We Need Inheritance?

💀 Because we're lazy and don't want to rewrite code.
Instead of writing the same shit again, we inherit it from another class.

2. Basic Inheritance – When Kids Copy Their Parents

🔥 Syntax:

```
class Parent:
    def display(self):
        return "This is the Parent class"

class Child(Parent):  # Child inherits Parent
    pass  # Child gets everything from Parent

c = Child()
print(c.display())  # Works fine! Child uses Parent's method.
```

Explanation:

Child(Parent) → The Child class inherits all the methods of Parent.

No need to rewrite display(), because it's already in Parent.

💀 This is like a kid copying homework from their dad's old notebooks.

3. Overriding – When Kids Think They're Smarter Than Parents 💀

🔥 If a child class doesn't like a method from the parent, it can override it.

```
class Parent:
    def display(self):
        return "This is the Parent class"

class Child(Parent):
    def display(self):  # Overriding the method
        return "This is the Child class"

c = Child()
print(c.display())  # Prints: "This is the Child class"
```

💀 Child says: "F*ck you, Dad! I'll do it my way!"

4. super() – When Kids Still Need Help From Parents 🤡

🔥 Use super() to call the parent's method inside the child class.

```
class Parent:
    def display(self):
        return "This is the Parent class"

class Child(Parent):
    def display(self):
        return super().display() + " + and Child class"

c = Child()
print(c.display())
# Output: "This is the Parent class + and Child class"
```

💀 This is like a rebellious teen asking for money after saying ‘I don’t need you.’

5. Types of Inheritance – Because Life Ain’t Simple 💀

🔥 Python has 5 types of inheritance:

a) Single Inheritance – Basic Copy-Paste

```
class Parent:
    def func1(self):
        return "Parent function"

class Child(Parent):
    def func2(self):
        return "Child function"

c = Child()
print(c.func1())  # Works fine
print(c.func2())  # Works fine
```

💀 Simple AF. Child gets everything from Parent.

b) Multiple Inheritance – When Kids Have Two Parents 💀

```
class Parent1:
    def func1(self):
        return "Function from Parent1"

class Parent2:
    def func2(self):
        return "Function from Parent2"

class Child(Parent1, Parent2):  # Inheriting from 2 parents
    pass

c = Child()
print(c.func1())  # Works fine
print(c.func2())  # Works fine
```

💀 This is like a kid getting genes from two different families.

c) Multilevel Inheritance – Grandparents Be Like 🤡

```
class GrandParent:
    def func1(self):
        return "Grandparent function"

class Parent(GrandParent):
    def func2(self):
        return "Parent function"

class Child(Parent):
    def func3(self):
        return "Child function"

c = Child()
print(c.func1())  # Inherited from Grandparent
print(c.func2())  # Inherited from Parent
print(c.func3())  # From Child
```

💀 Grandparents → Parents → Kids. The inheritance chain continues.

d) Hierarchical Inheritance – When Siblings Exist 💀

```
class Parent:
    def func1(self):
        return "Parent function"

class Child1(Parent):
    pass

class Child2(Parent):
    pass

c1 = Child1()
c2 = Child2()
print(c1.func1())  # Works fine
print(c2.func1())  # Works fine
```

💀 Multiple kids, one parent. Just like a poor Indian household.

e) Hybrid Inheritance – When Shit Gets Complicated 🤡

🔥 It's a mix of everything above.

```
class Parent:
    def func1(self):
        return "Parent function"

class Child1(Parent):
    pass

class Child2(Parent):
    pass

class GrandChild(Child1, Child2):
    pass

gc = GrandChild()
print(gc.func1())  # Works fine
```

💀 At this point, it's a whole family tree gone wrong.

Moral of the Story 💀

Inheritance = Copying parent's traits, like laziness and stupidity.

Overriding = Kids rebelling and doing things their way.

super() = Kids still needing help despite acting tough.

Types of inheritance = How complicated family trees can get.

🔥 Next up: Polymorphism – How One Thing Can Have Multiple Forms 💀

Chapter 26: Polymorphism – When One Thing Can Be Many Things 💀

Alright, dumbasses, today we talk about Polymorphism, which is just a fancy ass word for "One thing, multiple forms."

💀 Example to help your dirty minds understand:
A girl from 🟧⬛ can be:

A waifu for lonely weebs 🤡

A slut for thirsty simps 💀

A normal girl (but only when she's not in videos)

🔥 SAME GIRL, DIFFERENT ROLES. That's polymorphism.

1. What the F*ck is Polymorphism?

Polymorphism means one function/method can work differently for different objects.

💀 Real-life example:

A knife can cut vegetables or stab a simp 💀

A car can be a race car or a slow-ass taxi

A human can be a hardworking coder or a lazy ass watching 🟧⬛

🔥 In Python, this means:

Methods with the same name but different behavior.

A function can work on different objects.

2. Method Overriding – When Kids Change the Rules

💀 Polymorphism happens when a child class changes the parent's method.
🔥 Example:

```
class Girl:
    def role(self):
        return "I'm a normal girl"

class Waifu(Girl):
    def role(self):
        return "I'm a waifu, UwU"

class Slut(Girl):
    def role(self):
        return "I'm here for some action, daddy 💀"

g1 = Girl()
g2 = Waifu()
g3 = Slut()

print(g1.role())  # Normal girl
print(g2.role())  # Waifu
print(g3.role())  # Slut
```

💀 SAME function role(), DIFFERENT OUTPUTS based on the class.

3. Method Overloading – When You Can Do It in Multiple Ways 🤡

💀 Python doesn't support true method overloading, but we can fake it.

🔥 Example:

```
class Video:
    def play(self, speed=1):
        return f"Playing at {speed}x speed"

v = Video()
print(v.play())       # Default speed
print(v.play(2))      # Fast forward
print(v.play(0.5))    # Slow-mo action 💀
```

💀 Same method play(), different ways to use it.

4. Operator Overloading – When + Means More Than Just Addition 💀

🔥 Example:

```
class Girl:
    def __init__(self, name):
        self.name = name

    def __add__(self, other):
        return f"{self.name} and {other.name} are now a ship 💀"

g1 = Girl("Lisa")
g2 = Girl("Jennie")

print(g1 + g2)  # Lisa and Jennie are now a ship 💀
```

💀 We just made the + operator work with objects.

Moral of the Story 💀

🔥 Polymorphism = One thing, many forms.

Method Overriding = Changing parent's method like a rebellious teen.

Method Overloading = Using the same method in different ways.

Operator Overloading = Making +, -, *, etc., work with custom objects.

💀 Next up: Abstract Classes – When Parents Force You to Follow Rules.

Chapter 27: Abstract Classes – When Parents Force You to Follow Rules 💀

Alright, you slow-brained morons, today we're talking about Abstract Classes, which are basically like strict-ass parents who force their kids to follow rules.

💀 Example:
Imagine you join 🟧⬛, but instead of doing whatever you want, they force you to follow a script. No creativity, no fun, just "Do what you're told, or get out." That's exactly what abstract classes do in Python.

1. What the F*ck is an Abstract Class?

🔥 An abstract class is a class that cannot be directly used.

It acts like a blueprint for other classes.

It has at least one abstract method (a method with no body).

Any child class must define those methods, or Python will throw a tantrum.

💀 Imagine 🟧⬛ but with rules:

Abstract Class: "Every girl in our industry must have an act() method."

Child Class (Slut, Waifu, etc.) must define act() in their own way.

🔥 Example:

```
from abc import ABC, abstractmethod

class Video(ABC):  # This is the strict-ass parent
    @abstractmethod
    def act(self):
        pass  # No implementation, just a rule

class Waifu(Video):
    def act(self):
        return "UwU senpai, notice me ~ 💀"

class Slut(Video):
    def act(self):
        return "Let's get straight to business, daddy 💀"

girl1 = Waifu()
girl2 = Slut()

print(girl1.act())  # UwU senpai, notice me ~ 💀
print(girl2.act())  # Let's get straight to business, daddy 💀
```

💀 Same rule, different performances.

2. Why the F*ck Do We Need This?

🔥 Real-world example:

A Porn Company (abstract class) sets the rule: "All actors must have an act() method."

Different actors (Waifu, Slut, MILF, Stepsis💀) will perform differently.

💀 Without an abstract class, some dumbass might forget to implement act(), and everything will break.

3. Making a Strict-Ass Parent Class

🔥 Let's create an abstract class that forces people to work.

```
class Parent(ABC):
    @abstractmethod
    def work(self):
        pass

class HardworkingChild(Parent):
    def work(self):
        return "I'm coding 24/7 and making money. 💰"
```

```
class LazyAss(Parent):
    def work(self):
        return "I just sit around watching 🟧⬛ all day. 💀"

child1 = HardworkingChild()
child2 = LazyAss()

print(child1.work())  # I'm coding 24/7 and making money. 💰
print(child2.work())  # I just sit around watching 🟧⬛ all day. 💀
```

💀 If a child class doesn't define work(), Python will slap it with an error.

4. Why You Should Care (But You Won't, Cuz You're Dumb 💀)

🔥 Abstract Classes help to:

Enforce rules (stop dumbasses from forgetting methods).

Make sure all child classes have necessary methods.

Make code structured & avoid dumb mistakes.

💀 It's like a parent making sure their kid doesn't turn into a jobless bum.

Moral of the Story 💀

🔥 Abstract Classes = Strict-ass Parents

You can't use them directly.

They force kids (child classes) to follow rules.

Without them, dumbasses will break the system.

💀 Next Up: Interfaces – When Parents Give You More Than One Job.

Chapter 28: Interfaces – When You Have to Juggle Multiple Jobs 💀

Alright, you brain-dead monkeys, if you thought abstract classes were strict, get ready for interfaces, the ultimate pain in the ass. 💀

🔥 An interface is like being forced to do multiple jobs at once.

Imagine a 🟧⬛ girl who has to be a Waifu, MILF, Stepsis, and a Nurse all at once.

She needs to follow the rules of each job separately.

💀 That's what interfaces do—they force a class to follow multiple sets of rules.

1. What the F*ck is an Interface?

🔥 An interface is a contract. If a class uses an interface, it must define every method in it.

In Python, interfaces are just abstract classes where every method is abstract.

A class can have multiple interfaces (which means more work, RIP 💀).

2. Why the F*ck Do We Need It?

💀 Imagine your lazy ass needs money, so you take three jobs:

OnlyFans model (explicit work) 💀

Gamer (streaming for simps) 🎮

Freelancer (pretending to be productive) 💻

🔥 Each job has different rules, and you must follow all of them.

That’s an interface.

3. Creating an Interface in Python (The Slavery Begins 💀)

🔥 Step 1: Create the Interfaces

```python
from abc import ABC, abstractmethod

class OnlyFansModel(ABC):
    @abstractmethod
    def post_content(self):
        pass

class Gamer(ABC):
    @abstractmethod
    def stream(self):
        pass

class Freelancer(ABC):
    @abstractmethod
    def code(self):
        pass
```

💀 Each job has its own "rules" (methods) that must be followed.

🔥 Step 2: Create a Class That Does All Three Jobs (The Ultimate Slave 💀)

```python
class BrokeAss(OnlyFansModel, Gamer, Freelancer):
    def post_content(self):
        return "Selling feet pics for rent money 💀"

    def stream(self):
        return "Playing games while simps donate their life savings 🎮"

    def code(self):
        return "Freelancing on Fiverr for $5 per project 💻"

worker = BrokeAss()

print(worker.post_content())  # Selling feet pics for rent money 💀
print(worker.stream())  # Playing games while simps donate their life savings 🎮
print(worker.code())  # Freelancing on Fiverr for $5 per project 💻
```

💀 This dude has to do everything, no choice.
🔥 That’s what interfaces do—they force classes to follow multiple rules.

4. Real-Life Example (More Ways to Ruin Your Life 💀)

🔥 Think of a PornHub star:

She might be an OnlyFans model (must post content).

A TikTok influencer (must dance like a clown).

A YouTuber (must fake personality for views).

💀 She has to follow all these rules at once.

🔥 Example in Code:

```
class PHStar(OnlyFansModel, Gamer, Freelancer):
    def post_content(self):
        return "New exclusive video dropping, subscribe for $9.99 💀"

    def stream(self):
        return "Reacting to my own videos on Twitch 🤡"

    def code(self):
        return "Pretending to know Python while making dumb courses 💀"

actress = PHStar()
print(actress.post_content())  # New exclusive video dropping, subscribe for $9.99 💀
print(actress.stream())  # Reacting to my own videos on Twitch 🤡
print(actress.code())  # Pretending to know Python while making dumb courses 💀
```

🔥 Now you get why interfaces are a pain in the ass?

If a class chooses an interface, it MUST follow all rules.

Moral of the Story 💀

🔥 Interfaces = Being Forced to Do Multiple Jobs

They define rules that must be followed.

A class can have multiple interfaces (which means multiple jobs).

Without interfaces, people would just be lazy and skip work.

💀 Next Up: Exception Handling – Because Some of You Are Already Errors.

Chapter 29: Exception Handling – Fixing Your Dumbass Mistakes 💀

Alright, listen up, you half-functioning NPCs. This chapter is about exception handling, a.k.a. what to do when your dumbass makes mistakes and your code breaks.

🔥 Think of exceptions like this:

You try to DM a PH model, and she leaves you on read → Error: RejectedException 💀

You try to divide by zero, and Python starts screaming at you → ZeroDivisionError 💀

You try to run a script without saving it, and PyCharm tells you to go f*ck yourself → FileNotFoundError 💀

1. What the F*ck is an Exception?

💀 Exceptions are errors that happen when the code is running.

Python is lazy as f*ck and doesn't check your mistakes before running.

So, when your dumbass does something stupid, it throws an exception and crashes.

🔥 Common Python Exceptions:

ZeroDivisionError → You tried to divide by zero like an idiot 💀

ValueError → You tried to convert "a" into an integer, what were you thinking? 🤡

TypeError → You tried to add a string to a number, bruh.

FileNotFoundError → You tried to open a file that doesn't even exist.

💀 Let's break this system before fixing it:

```
print(10 / 0)  # ZeroDivisionError
int("hello")  # ValueError
open("non_existent_file.txt")  # FileNotFoundError
```

🔥 Congratulations, your program just crashed.

2. How to Handle Your Dumbass Mistakes (Try-Except) 💀

Python gives you a second chance at life with try-except blocks.

Try → The code that might fail

Except → What to do when it fails

💀 Example:

```
try:
    print(10 / 0)  # Bro is trying to divide by zero 💀
except ZeroDivisionError:
    print("Bruh, you can't divide by zero. Are you okay? 💀")
```

🔥 Output:

```
Bruh, you can't divide by zero. Are you okay? 💀
```

🔥 Saved your ass from a crash.

3. Handling Multiple Exceptions (Because You Suck at Coding) 💀

Sometimes, your dumbass makes more than one mistake at once. Python lets you catch multiple exceptions.

💀 Example:

```
try:
    x = int(input("Enter a number: "))  # What if the user types "ass" instead? 💀
    print(10 / x)  # What if they type 0? 💀
except ZeroDivisionError:
    print("Bruh, you really tried dividing by zero again? 💀")
except ValueError:
    print("Bro, enter a f*cking number, not alphabets. 🤡")
```

🔥 Test it by entering:

0 → "Bruh, you really tried dividing by zero again? 💀"

"ass" → "Bro, enter a f*cking number, not alphabets. 🤡"

4. Finally Block (When You Need to Clean Up the Mess) 💀

🔥 Finally is like cleaning up after a f*cked-up night.

No matter what happens, it runs.

💀 Example:

```
try:
    print("Trying to DM a PH model... 💀")
    raise Exception("She left you on read 💀")  # Manually raising an error
except Exception as e:
    print(f"Error: {e}")
finally:
    print("Bro, move on. She was never yours. 💀")
```

🔥 Output:

```
Trying to DM a PH model... 💀
Error: She left you on read 💀
Bro, move on. She was never yours. 💀
```

🔥 Moral of the story?

Even if things go wrong, life goes on.

Finally always runs, no matter what happens.

5. Raising Your Own Errors (When You Want to Make Life Worse) 💀

🔥 You can make Python cry by manually throwing errors.

💀 Example:

```
age = int(input("Enter your age: "))

if age < 18:
    raise Exception("You are too young for this website. 💀")
else:
    print("Welcome to PH Premium. 💀")
```

🔥 Enter 17:

```
Exception: You are too young for this website. 💀
```

💀 Bro, Python just kicked you out.

Moral of the Story 💀

🔥 Exceptions = F*ck-ups in your code

Python throws an exception when something goes wrong.

Try-except saves your dumbass from crashing the whole program.

Finally always runs, like your mom's disappointment in you.

Raise lets you manually throw errors, just to ruin someone's day.

💀 Next Chapter: Over Loading– Because Your Code Should Work Harder Than You Do.

Chapter 30: Overloading – When One PH Video Ain't Enough 💀

Alright, dumbasses, today's topic is overloading—a concept that lets you use the same function name in different ways. Just like how PH girls can be a waifu, a dominatrix, or a school teacher in different videos. 💀

🔥 What is Overloading?

Using the same function name but with different inputs or behaviors.

Real-life example:

A waifu in anime → Cute, wholesome, "Ara Ara~"

A waifu in 🟧⬛ → Not wholesome anymore 💀

Same name, different roles.

1. Function Overloading (One Name, Multiple Styles) 💀

🔥 In Python, function overloading is kinda fake. Unlike C++ and Java, Python doesn't support it directly, but we can fake it like an OF model pretending to be "innocent." 💀

💀 Example:

```
class PH:
    def recommend(self, category=None):
        if category == "MILF":
            print("Recommending: Step-Mom caught you coding 💀")
        elif category == "Teen":
            print("Recommending: Schoolgirl learns Python 💀")
        else:
            print("Recommending: Default PH algorithm (you're doomed) 💀")

ph = PH()
ph.recommend()  # Default
ph.recommend("MILF")  # MILF category
ph.recommend("Teen")  # Teen category
```

🔥 Output:

```
Recommending: Default PH algorithm (you're doomed) 💀
Recommending: Step-Mom caught you coding 💀
Recommending: Schoolgirl learns Python 💀
```

🔥 Same function name (recommend), but different results based on input.

2. Operator Overloading (Making + Do Some Freaky Stuff) 💀

🔥 In Python, even operators like +, -, and * can be overloaded.
🔥 Basically, + isn't just for numbers—it can do whatever the f*ck you want.

💀 Example (Adding Two PH Accounts 💀):

```python
class PHAccount:
    def __init__(self, username, videos_watched):
        self.username = username
        self.videos_watched = videos_watched

    def __add__(self, other):
        return PHAccount(self.username + " & " + other.username,
                         self.videos_watched + other.videos_watched)

    def __str__(self):
        return f"Account: {self.username}, Videos Watched: {self.videos_watched}"

acc1 = PHAccount("HornyBoy69", 420)
acc2 = PHAccount("StepBro", 690)

merged_acc = acc1 + acc2
print(merged_acc)
```

🔥 Output:

```
Account: HornyBoy69 & StepBro, Videos Watched: 1110
```

🔥 Congrats, you just merged two PH accounts. 💀

🔥 What Happened Here?

The + operator was overloaded to combine usernames and add watch history.

Normally, + is for numbers, but here we made it work for PH addiction tracking. 💀

3. Method Overriding (When You Change the Family Tradition) 💀

🔥 Method Overriding happens when a child class rewrites its parent's method.
💀 Like how a PH girl's daughter might refuse to follow the "family business" and become a doctor instead.

💀 Example:

```python
class Parent:
    def career(self):
        print("I am a software engineer.")

class PHModel(Parent):
    def career(self):
        print("I am a top 0.1% creator on OF 💀")

p1 = Parent()
p2 = PHModel()

p1.career()  # Normal Parent
p2.career()  # PH Model rewriting history 💀
```

🔥 Output:

```
I am a software engineer.
I am a top 0.1% creator on OF 💀
```

🔥 The child class (PHModel) rewrote the parent's method.

Moral of the Story 💀

🔥 Overloading = Same name, different actions.

Function Overloading → Same function, different inputs.

Operator Overloading → Making +, -, and * do crazy sh*t.

Method Overriding → Rewriting history like PH models rewriting their past.

💀 Next Chapter: Multi-Threading – How to Watch PH in 20 Tabs at Once.

Chapter 31: Multi-Threading – Watching PH in 20 Tabs at Once 💀

Alright, dumbasses, today we are talking about multi-threading—basically, making Python do multiple tasks at the same time. 💀
Just like how your degenerate ass has 20 PH tabs open at once while trying to "study." 💀

1. WTF is Multi-Threading?

🔥 Single-threading = Watching PH in one tab (boring as f*ck).
🔥 Multi-threading = Watching PH in multiple tabs at the same time (high risk, high reward 💀).

💀 Example:

You are coding while watching PH and downloading an "educational" video at the same time.

Your PC fan starts screaming like a PH actress.

That's multi-threading. 💀

2. Creating Multiple Threads (Multiple PH Tabs Running Simultaneously 💀)

🔥 In Python, we use the threading module to create multiple threads.
🔥 Basically, making Python work like PH premium. 💀

💀 Example:

```
import threading
import time

def watch_ph(tab):
    print(f"Tab {tab}: Opening PH...💀")
    time.sleep(2)
    print(f"Tab {tab}: Video loaded.💀")
    time.sleep(3)
    print(f"Tab {tab}: Video finished. Time to "clear history" 💀\n")

# Creating multiple PH tabs
threads = []
for i in range(5):  # Open 5 PH tabs
    t = threading.Thread(target=watch_ph, args=(i+1,))
    threads.append(t)
    t.start()

# Wait for all threads to finish
for t in threads:
    t.join()

print("All PH tabs closed. Now go touch grass. 💀")
```

🔥 Output:

```
Tab 1: Opening PH...💀
Tab 2: Opening PH...💀
Tab 3: Opening PH...💀
Tab 4: Opening PH...💀
Tab 5: Opening PH...💀
...
Tab 1: Video finished. Time to "clear history" 💀
...
All PH tabs closed. Now go touch grass. 💀
```

🔥 Each PH tab is running in a separate thread, meaning they open and close independently. 💀

3. Multi-Threading vs Multi-Processing (The Real Battle 💀)

🔥 Multi-threading = Multiple PH tabs running at the same time on the same CPU.
🔥 Multi-processing = Multiple PH videos running on different CPUs.

💀 Example (Multi-Processing PH Addiction 💀):

```python
import multiprocessing
import time

def watch_ph_process(tab):
    print(f"Process {tab}: Watching PH on another CPU 💀")
    time.sleep(3)
    print(f"Process {tab}: Done. History deleted. 💀\n")

if __name__ == "__main__":
    processes = []
    for i in range(5):  # Open 5 PH processes
        p = multiprocessing.Process(target=watch_ph_process, args=(i+1,))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

    print("All PH processes finished. FBI is watching. 💀")
```

🔥 Output:

```
Process 1: Watching PH on another CPU 💀
Process 2: Watching PH on another CPU 💀
Process 3: Watching PH on another CPU 💀
Process 4: Watching PH on another CPU 💀
Process 5: Watching PH on another CPU 💀
...
All PH processes finished. FBI is watching. 💀
```

🔥 Multi-processing uses multiple CPU cores, making it even faster.
💀 Basically, if your PC is overheating, now you know why.

Moral of the Story 💀

Single-threading = Watching PH in one tab (boring as f*ck).

Multi-threading = Watching PH in 20 tabs (risk of getting caught 💀).

Multi-processing = Watching PH on 5 different devices (FBI OPEN UP 💀).

💀 Next Chapter: Exception Handling – How to Not Get Caught Watching PH.

Again cuz WHY NOT💀

DOWN THERES SOME REAL ASS NEWS

°°°

🎓 Official Certificate of Dumbassary – Now LEGIT & Printable 💀

Alright, morons, this time it's real. No more fake promises like last time when you were expecting a certificate and just got your ass roasted instead. 💀

📜 The Certificate Process

1️⃣ Complete this book – Don't just scroll past like a dumbass. Actually learn something.
2️⃣ Mail me at 📩 dwipbiswas@yahoo.com with the subject: "I Finished Python for Losers 2"
3️⃣ You'll receive a test link – But hold on, this isn't some easy-ass quiz. You need to score at least 90% or else you're officially a dummy.
4️⃣ Test your skills – The test will be real-time with no retry attempts, so if you fail, too bad, go back to chapter 1, dumbass. 💀
5️⃣ After passing, you'll be asked for some details – Basic stuff like your name (so your dumbass can flex the certificate), email, and country (to see how many dumbasses are worldwide).
6️⃣ Wait a few days (or a week) – Because I'm lazy as f*ck and your certificate isn't my life's priority.
7️⃣ Get your certificate – A real, printable PDF that you can flex or use as toilet paper, your choice.

⚠️ WARNING ⚠️

If you fail the test, no certificate for your dumbass. 💀

If you try to cheat, the system will know, and your certificate will be replaced with a "Certified Dumbass" tag. 💀

If you don't apply, don't come crying later that you didn't get sh*t.

If you actually pass, congrats, you're NOT a dummy… barely.

So, if you really want to prove your worth, get through this book, take the test, and earn that damn certificate. Otherwise, enjoy being a Certified Dumbass for life. 💀

°°°

📢 Listen Up, You Absolute Buffoon 💀

Alright, so down below, you'll find some test preparation materials that might help you pass the upcoming "Python for Losers 2 - The Roastening" final exam. If you actually want the "Legit Certificate of Big Brain" and not the "Certified Dumbass" award, then you better start studying this shit.

I've handpicked the most important topics, some easy, medium, and hard questions, and even a bonus challenge to see if you actually have a functioning brain.

If you think you're smart enough, go through these materials, revise the topics, and then email me at dwipbiswas@yahoo.com once you're ready to take the real test. If you score below 90%, I'm sending you straight to remedial Python classes 💀.

Good luck. Or don't. I honestly don't care 💀🔥.

🟢 Easy Questions (100 Total) – For the Brain-Dead Beginners 💀

Alright, dumbasses, these are the easy questions. If you struggle with these, just close this book and start selling potatoes instead 🤡.

Q1 - Q10: Basic Syntax & Printing (If you get these wrong, retire from coding 💀)

1. What function is used to print something in Python?

2. How do you write a comment in Python?

3. What is the correct way to declare a variable?

4. What does print(5 + "5") do?

5. How do you take user input in Python?

6. What will print(2 ** 3) output?

7. What’s the difference between = and ==?

8. What will print(type(3.14)) output?

9. How do you print multiple items in one print() statement?

10. What does print("Hello" * 3) output?

Q11 - Q20: Data Types & Variables

11. Name the four primary data types in Python.

12. What is the data type of True?

13. What does type(10.0) return?

14. What is the output of print(5 // 2)?

15. How do you check the length of a string?

16. What is the difference between int() and float()?

17. What will str(123) + "456" output?

18. What does len([1, 2, 3]) return?

19. How do you declare an empty dictionary?

20. What happens if you divide by zero in Python?

Q21 - Q30: Operators & Expressions

21. What is the difference between and and or?

22. What is 5 % 2?

23. What does not True evaluate to?

24. What is 5 * 2 == 10?

25. What does bool(0) return?

26. What does bool("False") return?

27. What does 3 + 2 * 2 output?

28. What does print(2 ** 3 ** 2) output?

29. What does 1 < 2 and 3 > 4 evaluate to?

30. What is the output of print(10 / 3)?

Q31 - Q40: Conditional Statements

31. How do you write an if statement?

32. What keyword is used for an alternative condition?

33. What is the difference between elif and else?

34. Can an if statement be empty?

35. What does if 0: evaluate to?

36. What does if "": evaluate to?

37. What does if []: evaluate to?

38. What does if None: evaluate to?

39. What happens if you don’t indent inside an if block?

40. What is pass used for?

Q41 - Q50: Loops

41. What are the two types of loops in Python?

42. How do you create a for loop?

43. How do you create a while loop?

44. What does break do?

45. What does continue do?

46. What does range(5) output?

47. What is for i in range(5, 10, 2)?

48. What is infinite looping?

49. How do you exit a loop early?

50. What is enumerate() used for?

Q51 - Q60: Functions

51. How do you define a function?

52. What is return used for?

53. Can a function return multiple values?

54. What is the difference between print() and return?

55. What is a parameter?

56. What is an argument?

57. Can a function call itself?

58. What is the default return value of a function?

59. What is the difference between def func(): and def func(): pass?

60. What happens if you forget return?

Q61 - Q70: Lists

61. How do you create a list?

62. What is the index of the first element in a list?

63. How do you access the last element of a list?

64. How do you add an item to a list?

65. How do you remove an item from a list?

66. What does list.append(5) do?

67. What does list.pop() do?

68. How do you check if an item exists in a list?

69. What does list.sort() do?

70. How do you reverse a list?

Q71 - Q80: Strings

71. How do you create a string?

72. What does len("hello") return?

73. How do you make a string uppercase?

74. How do you make a string lowercase?

75. How do you remove spaces from a string?

76. What is slicing in strings?

77. How do you replace a word in a string?

78. What does "hello".count("l") return?

79. How do you check if a string starts with a specific word?

80. How do you check if a string ends with a specific word?

Q81 - Q90: Dictionaries & Tuples

81. How do you create a dictionary?

82. How do you access a value in a dictionary?

83. How do you add a new key-value pair?

84. How do you delete a key from a dictionary?

85. What does dict.keys() do?

86. What does dict.values() do?

87. What does dict.items() return?

88. How do you check if a key exists in a dictionary?

89. How do you get the length of a dictionary?

90. What is a tuple?

Q91 - Q100: Miscellaneous (Random Dumbass Checks 💀)

91. What does None mean in Python?

92. What does type(None) return?

93. What is isinstance(5, int)?

94. How do you swap two variables in Python?

95. What is a lambda function?

96. What does global do?

97. What does nonlocal do?

98. What does dir(str) return?

99. What does help(print) do?

100. What is import this?

📢 If You Failed Even One Question...

💀 You should rethink your entire life. These were the easy ones. If you failed, just stop and go watch Cocomelon.

But if you aced it, congratulations, you're slightly less dumb. Now, get ready for the Medium-Level questions where I make sure your brain actually works.

📢 Attention, Dumbasses! 💀

If you think you can't solve these easy-ass questions, you got two options:

1️⃣ Mail your dumbassery to me at dwipbiswas@yahoo.com so I can personally roast your lack of brain cells. 💀🔥

2️⃣ Keep struggling in silence and risk getting called a certified dumbass when you fail the real test. 🤡👏💯

Choose wisely... or else your future in coding will be about as bright as a wet matchstick. 💀

📢 Ayo, Welcome to the Medium-Difficulty Section! 💀

Alright, you survived the easy ones (or maybe you cried and emailed me 💀). Now it's time to see if you actually got some brain cells left or if that was just luck.

Same rules apply:

If you don't know shit, send your sob story to dwipbiswas@yahoo.com 💀🔥.

If you skip too much, just know the real test won't be so kind 🤡.

If you get through this, congrats! You're slightly less of a dumbass.

Now get your 🟧⬛-watching ass to work before I start questioning your whole bloodline. 💀✋🍑

🔥 Medium-Difficulty Python Questions (1-50) 💀

1. Write a Python program to swap two numbers without using a third variable.

2. Explain the difference between is and == in Python.

3. How do you define a function with a default parameter in Python?

4. Write a Python function to find the factorial of a number using recursion.

5. Explain the difference between shallow copy and deep copy in Python.

6. Write a Python program to check if a given string is a palindrome.

7. How do you sort a dictionary by values in Python?

8. What is Python's zip() function, and how is it used?

9. How can you merge two dictionaries in Python 3.5+?

10. Write a Python program to find the second-largest number in a list.

11. Explain how Python's garbage collection works.

12. What is the difference between append() and extend() in a list?

13. Write a function that checks whether a given number is prime.

14. How do you reverse a list in Python without using reverse() or slicing?

15. What is the use of self in Python classes?

16. What will be the output of bool([]), bool({}), and bool(set())?

17. How does Python handle memory management for mutable and immutable objects?

18. What are Python decorators, and how do they work?

19. How can you check if two strings are anagrams of each other in Python?

20. What is Duck Typing in Python? Give an example.

21. Explain the difference between a generator and a list comprehension.

22. Write a Python program to find the common elements between two lists.

23. How do you handle file exceptions in Python? Give an example.

24. Explain Python's Global Interpreter Lock (GIL).

25. What’s the difference between classmethod and staticmethod?

26. Write a program to find the intersection of two sets without using & operator.

27. How can you remove duplicate values from a list while maintaining order?

28. Explain the difference between None and False in Python.

29. What will be the output of this code?

```
def add_to_list(val, lst=[]):
    lst.append(val)
    return lst

print(add_to_list(1))
print(add_to_list(2))
print(add_to_list(3, []))
print(add_to_list(4))
```

30. How do you implement a stack using a list in Python?

31. Explain how break, continue, and pass work in loops.

32. Write a Python program to convert a list of tuples into a dictionary.

33. What is list comprehension, and how does it work?

34. How can you flatten a nested list using recursion?

35. What’s the difference between pop(), remove(), and del in a list?

36. Write a Python program to generate the first n Fibonacci numbers using recursion.

37. Explain the difference between deepcopy and normal assignment (=) in Python.

38. What is the output of the following code?

```
x = 5
y = x
y += 3
print(x, y)
```

39. How can you use a lambda function inside map() to square all numbers in a list?

40. Write a function that returns the sum of digits of a given number.

41. How does Python’s try-except-finally block work?

42. What are namedtuples, and how are they different from normal tuples?

43. Explain the concept of monkey patching in Python.

44. How do you find the most frequent element in a list?

45. Write a Python program to sort a list of tuples based on the second value.

46. How do you create a private variable in a Python class?

47. What is the difference between __str__() and __repr__() in Python?

48. How do you check if a string contains only digits?

49. Write a Python program to count the occurrences of each character in a string.

50. What will be the output of this code?

```
def func(x, lst=[]):
    lst.append(x)
    return lst

print(func(10))
print(func(20))
print(func(30, []))
print(func(40))
```

📢 ***Declaration for the Dumbasses*** 📢

Alright, listen up, morons 💀. If you think you're too smart for this test, don't come crying when your brain short-circuits halfway through. Now, here's the deal:

🚨 ***You are ALLOWED to use Google*** 🚨
✔ ***ONLY if you are genuinely stuck.***
✔ ***ONLY if that topic is not covered in this book.***
✔ ***ONLY if your tiny brain cannot recall the answer after actually trying.***

But if you abuse Google like a desperate clown 🤡, just know that you're only fooling yourself. If you can't pass without cheating, don't even bother applying for the certificate—just go print "I am a certified dumbass" and slap it on your forehead.

Now, go on and cry through the questions 💀.

Here's the next 50 questions. If you fail these, just know that even a toaster has more processing power than your brain. 💀

51-100: Medium-Hard Level Python Questions

51. What is the difference between is and == in Python?

52. How does Python handle memory management?

53. What is the purpose of the with statement in file handling?

54. Explain the difference between mutable and immutable data types with examples.

55. What is the use of the enumerate() function?

56. Write a function to check if a number is prime.

57. How do you reverse a list without using .reverse()?

58. Explain how list comprehension works with an example.

59. How do you swap two variables in Python without using a third variable?

60. What is the difference between deep copy and shallow copy?

61. How do you handle exceptions in Python? Write an example.

62. What is the purpose of the finally block in exception handling?

63. Write a program to find the factorial of a number using recursion.

64. What are Python's built-in data types?

65. How do you create and use a lambda function?

66. Explain the use of map(), filter(), and reduce().

67. What is the output of print(bool([])) and why?

68. Write a program to count the occurrences of each word in a given text.

69. How do you merge two dictionaries in Python 3.9+?

70. What are Python generators? How do they differ from normal functions?

71. What does the zip() function do? Provide an example.

72. Explain the difference between global and nonlocal keywords.

73. What is duck typing in Python?

74. Explain method overloading in Python.

75. Write a class in Python with a constructor and a destructor.

76. How do you create a singleton class in Python?

77. What is the __call__ method in Python?

78. What are @staticmethod, @classmethod, and instance methods?

79. What are Python magic methods? Explain __str__() and __repr__().

80. How do you implement multiple inheritance in Python?

81. Write a program to find the second-largest element in a list.

82. What is a metaclass in Python?

83. Explain the difference between .py and .pyc files.

84. What is the difference between list, tuple, set, and dictionary?

85. Write a program to check if a string is a palindrome.

86. How do you generate random numbers in Python?

87. What is memoization? How do you implement it in Python?

88. Explain join() and split() with examples.

89. How do you convert a list into a set?

90. How can you remove duplicate elements from a list?

91. Explain the difference between pass, continue, and break.

92. What does self refer to in a class?

93. What are Python decorators? Write an example.

94. Explain the purpose of super() in Python.

95. How do you implement a queue in Python using a list?

96. What is the difference between a shallow copy and a deep copy in Python?

97. How do you handle circular imports in Python?

98. What is the purpose of the yield keyword?

99. Explain the use of try, except, else, and finally.

100. [SECRET QUESTION] If you get this one wrong, you will cry. What is the meaning of life? 🤡💀

You better get at least 90% correct, or don't even bother applying for the certificate. If you do fail, just know your entire 100-generation family tree is disappointed in you. 💀

🚨 WARNING: IMPOSSIBLE QUESTIONS AHEAD 🚨

If you thought the previous sections were hard, buckle up, because now we're entering hell mode. 💀 These questions will destroy your soul, question your life choices, and make you regret ever picking up this book.

Rules:

You cannot use Google. (If you do, may your keyboard break mid-search. 💀)

If you get any of these right, you might actually be built different.

If you fail, don't even think about applying for the certificate. Instead, go back and relearn everything like the dumbass you are.

Now, prepare to suffer. 🤡

Chapter: Clearing the Mess – The Shit You Didn't Understand 💀

Alright, dumbasses, before we move on to the "Impossible Questions That Will End Your Bloodline," let's take a moment to clean up the mess. 💀 Throughout this book, I might have skipped, rushed, or completely ignored some things because I assumed you had at least two brain cells to figure them out. But judging by the number of people who will probably fail the test, I was clearly too optimistic. 🤡

So here's the chapter where I go over everything that might have left you confused, crying, or questioning your life choices.

1️⃣ What the Hell is __init__? (Constructors Recap)

This is Python's way of saying

> "Hey dumbass, every time you create an object, this function will run automatically."

It's called a constructor and is mostly used to initialize variables when you make an object. Here's a simple example to refresh your rotting memory:

```
class Waifu:
    def __init__(self, name, thickness):
        self.name = name
        self.thickness = thickness

anime_girl = Waifu("Zero Two", "Thicc")
print(anime_girl.name)  # Output: Zero Two
```

Translation: Every time you create a Waifu, you must tell Python her name and her thickness level or it won't work. 💀

2️⃣ Why Do We Use self Like a Dumbass?

You might be wondering,

> "Why do I keep typing self in every class method like an NPC?"

Here's the truth: self is just a fancy way of saying "I am referring to this specific object."

💀 Without self, your variables would be homeless. 💀

Example:

```
class Simp:
    def __init__(self, waifu_name):
        self.waifu_name = waifu_name  # Belongs to this object ONLY

senpai = Simp("Rem")
print(senpai.waifu_name)  # Output: Rem
```

Without self, Python wouldn't know what the hell you're talking about and your waifu would be lost in the void. Tragic. 💀

3️⃣ *args and **kwargs – The Mysterious Asterisks

Python lets you be lazy as hell by allowing multiple arguments in a function.
But then, some idiot asks,

> "Bro, what if I don't know how many arguments there are?"

Python: "Bet."

*args → Takes any number of arguments

**kwargs → Takes any number of keyword arguments

Example:

```
def my_function(*args, **kwargs):
    print("Arguments:", args)
    print("Keyword Arguments:", kwargs)

my_function(69, 420, waifu="Asuna", game="GTA 6")
```

Output:

```
Arguments: (69, 420)
Keyword Arguments: {'waifu': 'Asuna', 'game': 'GTA 6'}
```

Translation: Python doesn't care how many arguments you throw at it, it'll catch them all. 🔥

4️⃣ The Difference Between is and == (You Probably Fked This Up)**

Common Mistake:
People think is and == are the same thing. They're not. 💀

== → Checks if values are the same

is → Checks if the objects themselves are literally the same in memory

Example:

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(a == b)  # True (Values are the same)
print(a is b)  # False (Different objects in memory)
print(a is c)  # True (Same object)
```

If you use is when you should use ==, may your code explode. 💀

5️⃣ Lambda Functions – For People Who Hate Typing

A lambda function is just a shortened version of a normal function.

Normal function:

```
def add(x, y):
    return x + y
print(add(2, 3))  # Output: 5
```

Lambda version (One-liner):

```
add = lambda x, y: x + y
print(add(2, 3))  # Output: 5
```

Use it when you’re too lazy to write a full function.

6️⃣ Decorators – Python's Way of Hacking Functions

You might have seen this weird @something before functions. That’s a decorator—it’s just a way to modify functions without actually changing them.

Example:

```
def add_power(func):
    def wrapper():
        print("Boosting your function...")
        func()
    return wrapper

@add_power
def normal_function():
    print("This function is now stronger!")

normal_function()
```

Output:

```
Boosting your function...
This function is now stronger!
```

💀 Python just lets you slap power-ups onto functions. 💀

7️⃣ Exception Handling – Because You’re Dumb and Will Break Your Code

Whenever your code crashes and burns, exception handling is there to save your dumbass.

Example:

```
try:
    x = 1 / 0  # Bruh, you can't divide by zero
except ZeroDivisionError:
    print("Nice try, Einstein.")
```

If you don’t use try-except, your program will just die instantly. 💀

8️⃣ File Handling – Saving Shit Like a Pro

If you still don’t know how to read and write files in Python, you deserve to fail the test.

Basic Example:

```
with open("myfile.txt", "w") as f:
    f.write("Hello, morons!")

with open("myfile.txt", "r") as f:
    print(f.read())  # Output: Hello, morons!
```

💀 That’s it. Just open, read/write, and close. Stop overcomplicating it. 💀

More Unexplained Shit – The Final Purge 💀

Alright, dumbasses, we're NOT done yet. I know some of you are still clueless as hell about certain things, so before we move on to the soul-crushing, spirit-breaking, impossible mode questions, let's erase all possible excuses you could have left.

This is the last chance to fix the gaps in your rotten knowledge before I throw you into the exam hall to suffer. 🤡

1️⃣ break, continue, and pass – The Three Weirdos

These three ruin everyone's life at some point, so let's break it down.

break → Yeets out of the loop completely

continue → Skips the current iteration and moves to the next

pass → Does literally nothing but keeps Python from crying

Example:

```
for i in range(5):
    if i == 2:
        break  # Stops the loop at 2
    print(i)
```

Output:

```
0
1
```

💀 Loop died at 2. 💀

```
for i in range(5):
    if i == 2:
        continue  # Skips 2 and moves on
    print(i)
```

Output:

```
0
1
3
4
```

💀 Loop ignored 2 and kept going. 💀

```
for i in range(5):
    if i == 2:
        pass  # Does nothing, just a placeholder
    print(i)
```

Output:

```
0
1
2
3
4
```

💀 Loop acted like nothing happened. 💀

2️⃣ Mutable vs Immutable – Why You Can't Change Some Things

Mutable = Can be changed
Immutable = Can't be changed

Examples:

Mutable: Lists, Dictionaries, Sets

Immutable: Strings, Tuples, Integers

💀 If you try to change an immutable object, Python will slap you. 💀

```
x = "Hello"
x[0] = "Y"  # 💀 This will crash
```

💀 Strings are immutable, meaning you CAN'T modify them directly.

3️⃣ Difference Between List, Tuple, and Set (The Three Stooges)

List → Ordered, Mutable, Allows duplicates

Tuple → Ordered, Immutable, Allows duplicates

Set → Unordered, Mutable, No duplicates allowed

Example:

```
my_list = [1, 2, 3, 3]  # ✅ Can change & allows duplicates
my_tuple = (1, 2, 3, 3)  # ❌ Can't change but allows duplicates
my_set = {1, 2, 3, 3}  # ❌ No duplicates allowed, only {1, 2, 3}
```

💀 Use a list if you wanna modify data. Use a tuple if you wanna lock it. Use a set if you don't want duplicates. 💀

4️⃣ Difference Between del, remove(), and pop() – Deleting Like a Pro

💀 Python has way too many ways to delete things. 💀

del → Deletes by index or the entire object

remove() → Deletes by value

pop() → Deletes by index but returns the deleted value

Example:

```
my_list = [1, 2, 3, 4]

del my_list[1]  # Deletes index 1 (Value: 2)
print(my_list)  # Output: [1, 3, 4]

my_list.remove(3)  # Deletes value 3
print(my_list)  # Output: [1, 4]

x = my_list.pop(0)  # Deletes index 0 and stores it in x
print(x)  # Output: 1
```

💀 Use del when you know the index, remove() when you know the value, and pop() when you wanna delete & use the value later. 💀

5️⃣ Difference Between @staticmethod, @classmethod, and Instance Methods

Instance Methods → Default methods, need self

Class Methods → Use @classmethod and cls instead of self

Static Methods → Use @staticmethod, don't need self or cls

Example:

```
class Weeb:
    def instance_method(self):
        return "This is an instance method!"

    @classmethod
    def class_method(cls):
        return "This is a class method!"

    @staticmethod
    def static_method():
        return "This is a static method!"
```

💀 Use @staticmethod when you don't need self. Use @classmethod when you want to modify the class itself. 💀

6️⃣ The Difference Between deepcopy() and copy()

💀 If you don't know this, your copied objects might betray you. 💀

copy() → Shallow copy (changes in one affect the other)

deepcopy() → True deep copy (completely independent objects)

Example:

```
import copy

original = [[1, 2, 3], [4, 5, 6]]
shallow = copy.copy(original)
deep = copy.deepcopy(original)

shallow[0][0] = 999
print(original)  # 💀 Original changed because it was a shallow copy!

deep[0][0] = 888
print(original)  # ✅ Original stays the same because it was deep copied!
```

💀 If you wanna truly clone something, use deepcopy(). If you just wanna reference it, use copy(). 💀

7️⃣ The None Type – The Silent Killer

💀 If you ever see None in your output, you fked up somewhere.** 💀

```
def bad_function():
    x = 5  # No return statement, so it returns None by default

print(bad_function())  # Output: None
```

💀 If you forgot to return a value, Python will just hand you None like an L. 💀

8️⃣ The isinstance() Function – Stop Checking Types Like a Caveman

If you ever wrote:

```
if type(x) == int:
```

💀 You need help. Use isinstance() like a normal human. 💀

Example:

```
x = 69
if isinstance(x, int):
    print("Yes, this is an integer.")
```

💀 This is cleaner, better, faster, stronger. 💀

The Final Brainfuck – The Mystery of .py, .pyc, and Other Binary Bullsh*t 💀

Alright, morons, listen up.
We've covered almost everything, but I know some of you still have question marks floating over your heads like a glitchy NPC. So, before I throw you into the final boss fight of Impossible Questions™, we need to decode all the missing mysteries that I conveniently skipped before.

And trust me, this shit is deep. So strap in. 🚀

1️⃣ What's the Difference Between .py, .pyc, .pyo, and .pyw?

💀 Python has more file extensions than your IQ can handle. 💀

.py → The normal Python script file you write and run.

.pyc → The compiled version of a .py file (Python Compiled).

.pyo → The optimized version of .pyc (less debug info, faster).

.pyw → Windows-specific script that runs without opening a console.

Example:

```
python myscript.py  # Runs the normal Python script
python -O myscript.py  # Creates a .pyo file (optimized)
```

💀 TL;DR:

.py = Your code.

.pyc = Python converting your code to faster bytecode.

.pyo = Optimized compiled version.

.pyw = For Windows GUI apps, no console opens.

2️⃣ What the Hell is Bytecode (.pyc)?

When you run a Python script, it doesn't directly execute your .py file. Instead, Python converts it into a .pyc file (bytecode) and then runs it.

Why?
💀 Because Python is slow as hell, so it pre-compiles your code to make it faster. 💀

Example:

```
python -m py_compile myscript.py
```

This will generate __pycache__/myscript.cpython-XX.pyc inside a __pycache__ folder.

💀 Bytecode makes your code faster but isn't true machine code (C/C++ are still faster). 💀

3️⃣ What is the __pycache__ Folder?

💀 Ever seen a weird-ass folder called __pycache__ appear randomly? That's Python storing compiled .pyc files to speed up execution.

💀 Delete it if you want, Python will just recreate it like a zombie. 💀

4️⃣ Source Code vs Bytecode vs Machine Code – WTF is the Difference?

Source Code: The Python code you write (.py).

Bytecode: The translated version (.pyc), understood by Python's interpreter.

Machine Code: The final CPU-executable code (Python does NOT reach this step, unlike C++).

💀 TL;DR: Python never reaches full machine code, which is why it's slower than compiled languages like C++. 💀

5️⃣ What's the Deal with .whl Files?

💀 Ever installed a library and saw some weird .whl file?

It stands for Wheel, and it's just a pre-compiled Python package that installs faster than using raw .tar.gz files.

Example:

```
pip install numpy.whl  # Installs numpy faster
```

💀 If you don't care, just let pip handle it and move on. 💀

6️⃣ What is Python's pickle Module and Why is it Cursed?

Pickle is Python's way of storing objects in a binary format.
💀 But it's also a security nightmare because it can run arbitrary code. 💀

Example:

```
import pickle

data = {"name": "Dwip", "age": 15}
binary_data = pickle.dumps(data)  # Converts to binary

print(binary_data)
```

💀 NEVER unpickle data from an untrusted source. It can execute malware. 💀

7️⃣ What's the Deal with .pyz Files?

.pyz is a Python ZIP archive that contains multiple Python scripts packed together.

💀 Think of it like .zip, but it's for Python scripts instead. 💀

Example:

```
zipapp -c my_script.py -o packed_script.pyz
```

💀 You can run it like a normal script:

```
python packed_script.pyz
```

Final Declaration Before We End This Madness 💀

Alright, dumbasses, this is it.
We've crushed every remaining brain cell you had left.
At this point, if you're still confused, I honestly don't know what to tell you. 🤡

BUT—I have good news.

In the NEXT book, we will study ALL of this in extreme detail.
We'll go into how Python actually runs, optimizations, performance tuning, and how to master Python at a god-tier level.

So for now—
Forget about all this.
Focus on surviving the next chapter (The Impossible Questions™).

Good luck, dumbasses. You'll need it. 💀

🔥 ATTENTION, MORTALS 🔥

You have officially entered GOD-TIER QUESTION TERRITORY. 💀
This ain't no kiddie quiz, this is pure suffering.

💀 Rules of Engagement:

We only expect ONE correct answer. If you can solve just ONE, you pass as a survivor.

If you solve 10+, you are officially COOKED. 🔥🔥 (Seek help, you might be built different).

If you attempt ALL of them… you're not human. You're a Python demon.

💀 WARNING:

These questions are so brutal, some of them have no known solutions.

If you cry, that's normal.

If you rage quit, that's normal.

If you see code in your nightmares, that's normal.

Now, brace yourself. It's time. 🚀

🔥 THE IMPOSSIBLE SECTION BEGINS 🔥

Question 1:
Write a Python one-liner that can reverse any given string, convert it into binary, apply ROT13 encryption, decode it back, and finally return the original string. 💀

Question 2:
Create a Python program that can solve a 100x100 Sudoku grid in less than a second using pure Python (no external libraries). If your program takes more than 1 second, you are a failure. 💀

Question 3:
Write a Python script that can compress any 1GB file down to just 1KB and decompress it back with 100% accuracy. If you do this, congratulations, you have broken computer science and deserve a Nobel Prize. 🤡

Question 4:

Implement a Python program that can accurately predict the next 10 numbers in the Fibonacci sequence without using recursion, loops, or mathematical formulas. 💀 (Yes, figure it out, genius.)

Question 5:
Develop a Python AI that can write an entire book explaining why the universe exists, how it will end, and the meaning of life. 🤡 The AI must have original thoughts, no copying from other sources, and be 100% scientifically accurate.

💀 If you are still breathing after reading these… good luck.

🔥 MORE IMPOSSIBLE QUESTIONS (BUT LOOKING LIKE NIGHTMARES) 🔥

Question 6:
A quantum computer has a superposition of infinite states and you need to simulate it using Python on a 20-year-old PC with 512MB RAM. Write a fully functional quantum simulator that runs efficiently without crashing. 💀

Question 7:
You have a list of 1 billion numbers. Your task is to sort them instantly without using sorting algorithms, loops, or recursion. If your solution takes more than 0.0001 seconds, you should reconsider your life choices. 🤡

Question 8:
Write a Python program that accurately calculates the exact date and time of the next World War, using only random.randint(). If your answer is wrong, well… 💀

Question 9:
Develop a Python AI chatbot that can perfectly mimic your own personality, thoughts, and humor so well that even you can't tell if you're talking to yourself or the AI. If it works, congrats, you've just destroyed human existence. 🤡

Question 10:
You are given a JPEG image of a cat. Write a Python script that reconstructs the original cat in 3D, including its skeleton, fur texture, and personality traits. The AI must also predict its favorite food and what it would name its kittens. 💀

💀 If you're still alive, you deserve an award for even attempting to read this.

🔥 ACTUAL POSSIBLE QUESTIONS (BUT STILL GONNA MAKE YOU SWEAT) 🔥

Question 11:
Write a Python program to check if a given number is prime without using loops or recursion.

Question 12:
Create a function that takes a string as input and returns True if it is a palindrome, ignoring spaces, capitalization, and punctuation.

Question 13:
Write a Python script that reads a file, counts the number of words, lines, and characters, and prints the result.

Question 14:
Create a Python class called "BankAccount" with attributes balance and owner_name. Implement methods for deposit, withdrawal, and balance check.

Question 15:
Write a function that takes a list of numbers and returns the second largest number.

Question 16:
Implement a function that generates a Fibonacci sequence up to the Nth term using a generator.

Question 17:
Write a Python function that takes a list of numbers and removes duplicates while maintaining the order.

Question 18:
Create a function that takes a list of words and sorts them based on their lengths (shortest to longest).

Question 19:
Write a Python script to find the factorial of a number using both recursion and iteration.

Question 20:
Create a simple calculator program that supports addition, subtraction, multiplication, and division using a menu-driven approach.

💀 If you can solve these, you're actually doing something right for once. 🤡

🔥 WELCOME BACK TO HELL 🔥

(This time, if you even TRY to run these on your potato PC, say goodbye to your CPU, RAM, and possibly your entire existence.)

Question 21:
Write a Python program to brute-force every possible 256-bit AES encryption key and decrypt a given ciphertext. (Enjoy burning your GPU to ashes.)

Question 22:
Implement a Python-based neural network from scratch, train it on a 1-billion-line dataset, and make it classify whether a given number is prime. (Bro, you need a whole data center for this 💀.)

Question 23:
Simulate the entire Universe in Python, including all planets, black holes, dark matter, and quantum interactions. (NASA wants to know your location if you even attempt this 💀.)

Question 24:

Write a Python script that solves an NP-complete problem (like the Traveling Salesman Problem) using brute-force for a dataset of 100,000 cities. (Your CPU just screamed in pain.)

Question 25:
Use Python to mine 1 Bitcoin with only your CPU. (Enjoy waiting till the heat death of the universe.)

Question 26:
Write a function that reverses a string but takes exactly 100 years to complete execution, regardless of input. (A true test of patience 💀.)

Question 27:
Create a Python program that can accurately predict stock market movements with 100% accuracy. (Even ChatGPT is laughing at this one 💀.)

Question 28:
Write a Python function that finds all prime numbers up to 10^100 using only a single-threaded CPU with 2GB RAM. (Your PC just applied for retirement.)

Question 29:
Create a real-time simulation of Earth’s weather down to individual molecules, running on your laptop. (Congrats, you just built a supercomputer.)

Question 30:
Use Python to calculate pi up to 1 trillion digits using only integer math, without using any external libraries. (You’ll finish computing this by the time humans colonize Mars.)

💀 If you can even THINK of solving one of these, congrats, you're cooked beyond saving.

🔥 STORAGE NIGHTMARE QUESTIONS 🔥

(If your PC survives this, it deserves a fcking award.)

Question 31:
Write a Python script that creates a 1PB (petabyte) file filled with random data. (Enjoy filling up your entire storage and then some 💀.)

Question 32:
Create a program that reads a 100GB text file line by line and counts the occurrences of each word without crashing. (Your RAM just gave up.)

Question 33:
Write a Python script that stores and retrieves 10 million high-resolution images in a single file without using databases. (Your SSD is officially dead.)

Question 34:
Create a compressed file format in Python that can reduce any file's size by 99% without losing data. (If you do this, every cloud storage company will sue you 💀.)

Question 35:
Write a Python program that keeps appending data to a file until the storage is full, then deletes it and starts over. (Infinite suffering mode activated.)

Question 36:
Simulate a hard drive's read/write operations at the sector level in Python. (Congrats, you just built a primitive SSD firmware 💀.)

Question 37:
Create a program that can recover deleted files from a hard drive using Python. (FBI be watching now.)

Question 38:
Write a Python function that mimics a RAM disk, storing all file operations in memory instead of disk. (If you run this on 4GB RAM, your PC is cooked.)

Question 39:
Use Python to store a complete human genome sequence (3 billion base pairs) efficiently and retrieve any segment in O(1) time. (Bro just reinvented bioinformatics 💀.)

Question 40:
Create a Python script that reads and processes every single file on your PC in real time. (You'll either delete System32 by accident or become an elite hacker.)

💀 IF YOU CAN SOLVE ANY OF THESE, YOUR STORAGE DESERVES A FUNERAL.

🌐 INTERNET COOKING QUESTIONS 🍳💀

(If your ISP sees this, they might cut your connection 🤡.)

Question 41:
Write a Python script that downloads the entire Wikipedia and saves it offline. (Congrats, you just created your own internet.)

Question 42:
Create a Python bot that auto-replies to every tweet on Twitter (X) with a custom message. (Enjoy getting banned 💀.)

Question 43:

Write a Python script that sends 100,000 emails in 1 second. (If your email provider allows this, they are insane.)

Question 44:
Develop a Python script that can bypass rate limits and scrape 1 million web pages in 10 minutes. (FBI be knocking soon.)

Question 45:
Create a Python program that tracks the location of every device on your WiFi network in real time. (You just became the NSA 💀.)

Question 46:
Make a Python script that continuously pings a website until it crashes. (Hello DDoS attack 💀.)

Question 47:
Write a program that can auto-login to any website, fill forms, and submit them within milliseconds. (Bots be taking over.)

Question 48:
Create a Python script that downloads every image, video, and file from a website automatically. (Bro just invented an illegal archive.)

Question 49:
Develop a Python tool that sends fake traffic to a website to inflate visitor count. (Web admins be raging rn.)

Question 50:
Write a script that creates 1,000 social media accounts automatically and starts posting memes. (Enjoy getting shadowbanned 💀.)

💀 IF YOU RUN ANY OF THESE, YOUR INTERNET PROVIDER MIGHT CALL THE POLICE.

📢 DECLARATION: YOU'RE NOT COMPLETELY COOKED... YET 💀

Alright, you've survived 50 internet-cooking questions, and if your brain isn't completely fried yet, congrats. 🎉

But let's be real—not all of them were impossible. Some were actually really easy (yes, I threw in a few to make you feel smart 🤡). A few of these questions were totally doable if you actually paid attention to the book instead of crying.

Now, before you get cocky, remember: the nightmare ain't over yet. The next set of questions is going to be so cursed, even your brain's RAM will run out.

So if you somehow managed to solve more than 3-5 questions so far, you're either a genius or just using Google like a true hacker. Either way, buckle up. 💀

💀 QUESTIONS 51-70: WELCOME TO THE GPU-PARADOX ZONE 💀

Alright, now we're stepping into true GPU hell. These questions will test your brain, patience, and your electricity bill if you dare to run them. If you own a single RTX 4090, good luck. If you have multiple 4090s in parallel, you might actually have a chance. 🤡

51. The Quantum Sorting Paradox

You have a list of N elements, and you must sort them in O(1) time using quantum entanglement. Write a Python function that does this. (Yes, it's a paradox. No, you're not dreaming.)

52. Schrödinger's Code

Write a Python program that, when executed, has a 50% chance of deleting itself and a 50% chance of printing "I'm alive." It must be truly random (no fake random.seed(0) nonsense).

53. Infinite Loop Escape Challenge

Create an infinite loop in Python that actually ends at some point without using break, return, exit(), or sys.exit().

54. The Multiverse Simulation

Simulate multiple universes using multiprocessing, where each process represents a different timeline. Each timeline should evolve independently and occasionally collide with another timeline, merging their histories. (Yes, your CPU will scream.)

55. The 2D Ray-Tracing GPU Killer

Write a ray-tracing algorithm in Python that can render a basic 2D scene (e.g., spheres, light sources) using only NumPy. It must run on CPU or GPU and should be fast enough to generate an image in under 5 minutes on an RTX 4090.

56. Paradox of the Reversed Python

Write a Python program that, when executed, reverses itself, but when reversed and executed again, still produces the original output.

57. The Never-Ending Dictionary

Write a Python dictionary that keeps growing infinitely but never runs out of memory (Hint: Think of a generator function).

58. The Collatz GPU Annihilation

Write a high-performance parallelized implementation of the Collatz Conjecture that can process a billion numbers in under a minute using an RTX 4090.

59. The Infinite Compression Algorithm

Write a Python function that can compress any file to 1KB and then decompress it back to its original size without any loss of data. (Yeah, this is basically magic. Let's see how you handle it.)

60. AI That Writes a Better Python Book Than Me

Create an AI model (without using OpenAI APIs) that can read this book and generate a better version—more roasts included. 💀

61. The GPU Overclocking Script

Write a Python script that can automatically overclock an RTX 4090 to its absolute limit without causing a thermal meltdown.

62. The Paradoxical Function Call

Write a Python function that calls itself before it exists without causing a NameError.

63. The Absolute Zero Bug

Write a Python program where dividing by zero does NOT cause an error, and actually returns a meaningful result.

64. The Quantum Entanglement Sorting 2.0

Modify bubble sort so that it can sort a list in O(1) time by taking advantage of quantum superposition. (Just pretend Python supports quantum physics.)

65. The 4D Rendering Madness

Create a Python script that can render a 4D object (like a tesseract) on a 2D screen while allowing the user to rotate it in real time.

66. The Time-Travel Debugger

Write a debugging tool that allows a Python program to reverse time and undo a mistake without restarting the program.

67. The Anti-Lag Gaming Algorithm

Write a network latency prediction model in Python that can eliminate ping in online multiplayer games. (If you actually solve this, game companies will throw money at you.)

68. The Unstoppable Server

Create a Python web server that can never be stopped. Even if the process is killed, it must somehow respawn automatically.

69. The Impossible Paradox: A Function That Runs in Negative Time

Write a Python function that, when executed, returns its result before it even starts running.

70. The Crypto Mining Python One-Liner

Write a one-liner Python script that efficiently mines cryptocurrency using only the CPU.

🔥 DECLARATION: YOU'RE EITHER A GOD OR YOU'RE CRYING 🔥

Alright, you've reached question 70, and you should be sweating by now. If you actually attempted any of these, you either:

Own NASA's supercomputer

Have a deep love for suffering

Or have already given up and are crying in the corner.

Either way, good job making it this far. 💀 Now, if you're still alive, get ready for the next round—it's gonna get even worse. 🤡

💀 QUESTIONS 71-90: THE INTERNET'S MOST CHAOTIC PROGRAMMING QUESTIONS 💀

Alright, these 20 questions are straight from the depths of Reddit, Discord, and YouTube comments—memes, cursed programming ideas, and actual challenges. If you can find the original video or post, you might just get the answer. Otherwise, cope and seethe. 💀

71. The Python One-Liner Madness

Someone posted a 1-liner Python script that could multiply numbers faster than NumPy without using NumPy. Find it and explain how it works.

72. The 5MB Python Hello World

There's a way to make a simple 'Hello World' program in Python take up 5MB. Find out how and recreate it.

73. The Discord Bot That Deletes Itself

A Discord bot was created that, upon being run, deletes its own source code from the host PC. Write a similar bot in Python.

74. The YouTube Compression Mystery

A YouTube video claims to explain an algorithm that can compress any file into a 1-byte representation and decompress it perfectly. Find it and explain why it's fake (or real?).

75. The Unstoppable while True Loop

There's a Reddit post about a Python while-loop that continues running even after the program is terminated. Find it and figure out how it works.

76. The 0x0 Brainfuck

A C program exists where 0x0 is actually a valid instruction that runs code. Find it and explain why it works.

77. The Unreal Number Crash

Someone posted a Python program where simply entering 0.0000001 - 0.00000009 in an old version of Python crashed their entire PC. Find out why.

78. The Python ASCII Horror Program

A small ASCII-based Python script that, when run, starts acting like it's haunted—changing its own output every time you run it. Recreate it.

79. The Ultimate eval() Challenge

Find the most dangerous way someone has misused Python's eval() function on StackOverflow or Reddit. Recreate it (without bricking your PC).

80. The Impossible Bug That Wasn't a Bug

A developer posted on Twitter about a Python bug that made their program run 10x faster for no reason. Find the post and explain the real reason behind it.

81. The Game That Runs in a Discord Message

Someone made a full playable game inside a Discord message using only emojis. Find it and figure out how it works.

82. The Accidental int Overflow Time Bomb

A JavaScript function was found that breaks after 292 years due to an integer overflow. Find the original post and translate the concept into Python.

83. The Forbidden Python Keyword

There's a secret Python keyword that crashes the interpreter if used incorrectly. Find it.

84. The Browser Crypto Miner That Stole Electricity

A website was exposed for running a hidden JavaScript crypto miner that used visitors' GPUs without their consent. Find the original post and explain how it worked.

85. The 10-Year-Old Code That Still Runs

A 10-year-old GitHub repo contains legacy Python 2 code that still runs perfectly on modern Python. Find it and explain how it's possible.

86. The "Impossible" import Statement

A YouTube tutorial showed a way to import something that doesn't exist in Python. Find it and explain how it worked.

87. The AI That Solved the Impossible Chess Puzzle

A chess AI was trained to solve a puzzle no human could solve. Find the post and recreate the AI in Python.

88. The Discord GIF That Installs Malware

Someone posted a malicious GIF that could execute code on Discord users' PCs. Find the post and explain how it worked.

89. The print() Function That Takes 10 Minutes

A single print() statement in Python took 10 minutes to execute without loops or recursion. Find the original code.

90. The 8-Bit Game Hidden in a Webpage

A developer hid a full 8-bit game inside a website's console. Find it and figure out how to play it.

🔥 DECLARATION: YOUR SANITY IS ON THIN ICE 🔥

If you actually found some of these posts/videos and solved the questions, you are:
✅ Chronically online
✅ Good at searching shit up
✅ Probably an AI in disguise

If you couldn't solve anything, go touch grass, then come back. ☠️

📢 NEXT 10 QUESTIONS ARE COMING FROM ME PERSONALLY. GET READY.

🔥 DECLARATION OF THE FINAL TANTUM CHALLENGE 🔥

Welcome to the Final Challenge—the last test of sanity, where math no longer obeys logic, and physics might collapse if you actually solve it.

This section is beyond human comprehension. It contains 10 impossible questions, crafted with the sole purpose of breaking the mind of even the most powerful AI. If you can solve even one, you are officially cooked. If you solve 10, you either:

Hacked reality itself

Are a quantum computer in disguise

Need immediate therapy 💀

💀 RULES OF SURVIVAL 💀

1️⃣ You must submit at least one answer to pass. If you submit more than 10, you are beyond saving.
2️⃣ You may use Google, but Google may not save you. 💀
3️⃣ If you somehow find a Reddit or Discord post with the exact answer, you are in a simulation. Escape while you can.
4️⃣ If you mail us crying for help, we will simply respond with: "Tragicc ≠ Tragicc - S'har'vann." 💀

5️⃣ The final question is a Python-based paradox, designed to either prove Tantum exists or crash your PC.

At the end of this challenge, we expect two types of people:
✅ Those who survived (1+ correct answers)
💀 Those who never returned (0 answers or attempted all 10)

If you dare proceed, may Koanaomi have mercy on your brain. 🕊️💀

💀 THE TANTUM EQUATION CHALLENGE 💀

The End of a Universe, Calculated in Python.

🌌 THE MYSTERY OF TANTUM & KOANAOMI 🌌

In a cursed parallel reality, beyond our universe, exists Tantum—a single-layered, infinitely stretching void. But it follows rules.

At the heart of Tantum is Koanaomi, the stabilizing force holding everything together. Inside Koanaomi, four Kintershiis keep trying to fuse. When they do, it triggers Tragicc, the final event that ends Tantum.

The whole collapse is controlled by Tracii, a cursed mathematical value that is the sum of all digits in the total energy of the Big Bang and the Final Collapse.

Your task? Solve the final fate of Tantum using Python. 💀

📜 THE FINAL EQUATION TO SOLVE

1️⃣ Find Kintershii using this nightmare equation:

Kintershii = 33e926437 × √π × (78%203372749)2919e8298473

2️⃣ Find Tracii by summing all digits of this total energy:

Tracii = sum of digits of (Big Bang Energy + Final Collapse Energy)

3️⃣ Check if Tragicc collapses the universe:

Tragicc = Tragicc : Tragicc ≠ Tragicc - S'har'vann

If Tragicc ≠ Tragicc, then Koanaomi collapses, and Tantum is gone forever.

🔥 YOUR PYTHON CHALLENGE 🔥

1️⃣ Calculate Tracii, the sum of digits of (Big Bang Energy + Final Collapse Energy), using Python.

2️⃣ Compute Kintershii using:
Kintershii = 33e926437 × √π × (78%203372749)2919e8298473

3️⃣ Solve for Tragicc using:
Tragicc = Tragicc : Tragicc ≠ Tragicc - S'har'vann
If it holds, the universe collapses.

4️⃣ Find the Koanaomi Collapse Time by dividing the final energy of Tantum by Tracii.

5️⃣ If the fusion of all four Kintershiis releases infinite energy, prove why using Python.

6️⃣ Write a Python function that checks whether Tragicc can be computed within finite time.

7️⃣ The Schrödinger-Kintershii Paradox states that Kintershii can be both real and imaginary. Write a Python script to verify this using complex numbers.

8️⃣ Simulate the Big Bang Energy Balance by writing a Python script that checks if the total energy before and after the collapse remains the same.

9️⃣ Convert the Kintershii Equation into a recursive Python function and try running it. (If your system freezes, you win.)

🔟 The final challenge: Use Python to generate a random equation that, when solved, outputs "Tantum Exists."

💀 WARNING 💀

If your Python code outputs infinity (∞) or crashes, congrats—you just solved the end of a universe.

Good luck. Physics is watching. 🚀🔥

🔥 FINAL NOTE: WELCOME TO EXISTENTIAL CRISIS 🔥

You made it. Or maybe you didn't. At this point, it doesn't even matter.

Look back. Think about what you just did.
You spent hours, maybe days, trying to solve equations that shouldn't even exist.
You stared at meaningless symbols, convinced yourself they had logic, and fought against Tantum itself.

And now… what do you have? A brain full of pain and a GPU begging for mercy.

Was it worth it?
Did solving these questions make you feel smarter?
Or did they just expose the fragility of your own existence?

If you solved nothing, congrats, you're sane.
If you solved everything, you are no longer human.
And if you refuse to answer this, then you have entered the void—where answers no longer exist.

This book was never about teaching.
It was about revealing the limits of your mind.

Now go, take a deep breath, and ask yourself:

"Why did I even do this?"

💀 WAIT, YOU THOUGHT IT WAS OVER? THINK AGAIN.

Before you crawl back into your normal life, remember this: the test is still waiting.
Yeah, that REAL test, not some fake-ass "you'll get a certificate" promise like before.

You must score at least 90% to not be labeled a dummy.

If you think you can just guess your way through, good luck, genius.

If you fail… well, let's just say your brain might need a factory reset.

Once you're done suffering through the book, mail me about your completion.
You'll get a link to the test, and then? You wait.
It might take a few days, maybe a week—because let's be real, I'm lazy 💀—but if you survive, you get the legit certificate.

And don't even think about skipping it.
Because after all this, if you run away now... everything you just went through was for NOTHING.

📢 FINAL RECAP: NO CHEATING, NO ESCAPING

Alright, before you even think about running away or pulling some sneaky BS, here's everything you need to know about the test process:

📧 Contact & Submission

Once you've FINISHED the book (yeah, no skipping), email me at dwipbiswas@yahoo.com with the subject:

"I Survived (Barely) – Test Application"

In the email, mention:

Your full name (or whatever cursed alias you go by)

Proof you actually completed the book (I'll know if you're lying 💀)

Your reason for existing after all this suffering (optional, but I wanna see your coping strategies)

🛑 Anti-Cheat Measures (You Thought You Were Smart?)

GOOGLE? Allowed, but only if you're truly stuck. If I see copy-paste nonsense, you're instantly disqualified.

AI? Bro, if you use AI, just admit defeat. I'll detect it, and you'll be automatically labeled dummy level -999.

Sharing answers? Go ahead. The test shuffles everything, so your so-called "genius squad" is just gonna fail together 💀.

Retakes? Nope. One attempt. If you mess up, you're done. But…. But if I'm in a mood you may get a chance… AND I MEAN YOU MAY GET A CHANCE NIT SURE….

⏳ What Happens After the Test?

If you pass (90%+), you get your certificate—a REAL, PRINTABLE ONE.

If you fail? Well… you might wanna reconsider your life decisions.

Certificate delivery might take a week (or more) because I'm lazy, but you'll get it.

Now, no more excuses. Either face the test, or admit defeat.

-Signing off

Dwip

[illegible]