

Python

Programming Practice Problems: Course Overview

- - Syntax drills, logic challenges, and data structure exercises
- - Debugging practice and real-world mini-projects
- - For beginners, developers, and educators
- - Active problem solving with immediate feedback loops
- - Build core Python command and debugging habits



Why Structured Practice Beats **Passive Watching**

- > - Reading syntax vs. writing code that runs
- > - Build muscle memory: loops, conditionals, functions
- > - Feedback loop: run, read errors, debug, refactor
- > - Research-backed: active learning beats passive viewing
- > - Consistency wins: 20-30 min daily over marathon sessions



PASSIVE WATCHING

Information in, little out.



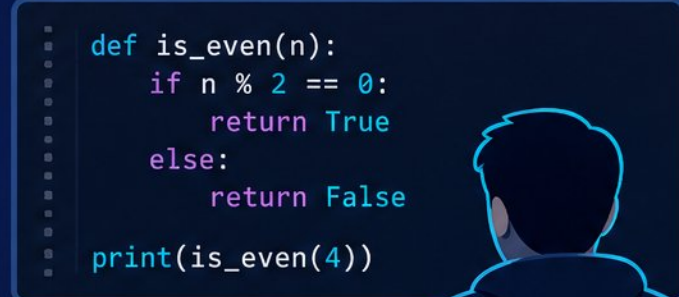
```
$ python script.py  
(no output)
```

VS



ACTIVE CODING

Practice out, skill in.



```
$ python script.py  
True
```



Active practice builds understanding, confidence, and problem-solving instincts.

Core Problem Types and a Realistic Difficulty Ramp



- Basics: variables, data types, input/output, simple calculations



- Logic: conditionals, loops, string and list manipulation



- Functions: modular decomposition and reusable code



- Intermediate: dictionaries, file handling, simple algorithms



- Progression: topic drills → combined challenges → mini-projects



Warm-Up Drills:

Syntax, Data Types, and I/O

- Master core types: integers, floats, strings, booleans
- Practice type conversion and arithmetic operations
- Build input and formatted output skills
- Quick drills with instant feedback build confidence
- Try: greeting generator, temperature converter, even/odd checker



PYTHON DATA TYPES

A quick specimen card

i

Integer

Whole numbers

42

f

Float

Decimal numbers

3.14

str

String

Text in quotes

"hello"

bool

Boolean

True or False

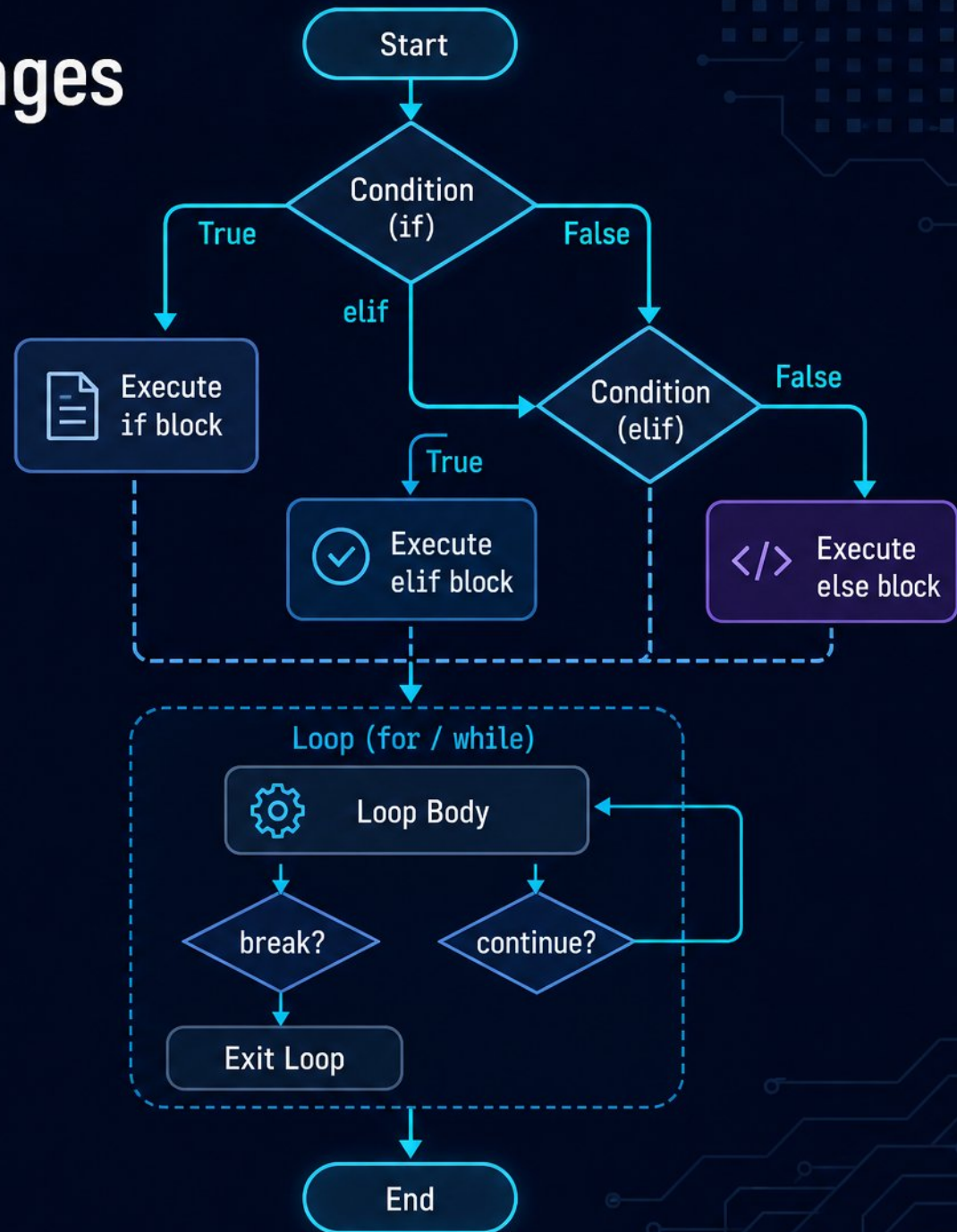
True



Understanding these core types is the foundation for writing clear and correct code.

Control Flow Challenges

- Master `if`, `elif`, `else` for decision logic
- Loop with `for` and `while` for iteration
- Control flow with `break` and `continue`
- Progress from simple checks to complex logic
- Practice: FizzBuzz, prime check, table generation



Data Structures in Action

- List basics: indexing, slicing, sorting, comprehensions
- Dictionaries for counting, grouping, and lookups
- Choose tuples for fixed data, sets for uniqueness
- Practice: word frequency, deduplication, intersections
- Pick the right structure for the problem



List

Ordered,
changeable

```
items = ['apple', 'banana',  
         'cherry']  
items[1]      # indexing  
items[0:2]    # slicing  
sorted(items) # sorting  
[x.upper() for x in items]
```



Dictionary

Key-value,
fast lookups

```
counts = {'apple': 1,  
         'banana': 2}  
counts['apple'] += 1    # count  
counts.get('pear', 0)  # lookup  
for k, v in counts.items():  
    pass                # iterate
```



Tuple

Ordered,
immutable

```
point = ('x', 'y')    # fixed data  
name, value = point   # unpack  
point[0]              # indexing  
# cannot modify  
# point[0] = 'z'      # not allowed
```



Set

Unordered,
unique items

```
letters = {'a', 'b', 'c'}  
letters.add('d')        # add  
letters.discard('a')    # remove  
other = {'c', 'd', 'e'}  
letters & other          # intersection
```

Functions, Scope, and Reusability



- Write functions with parameters and return values



- Understand scope to avoid side-effect bugs



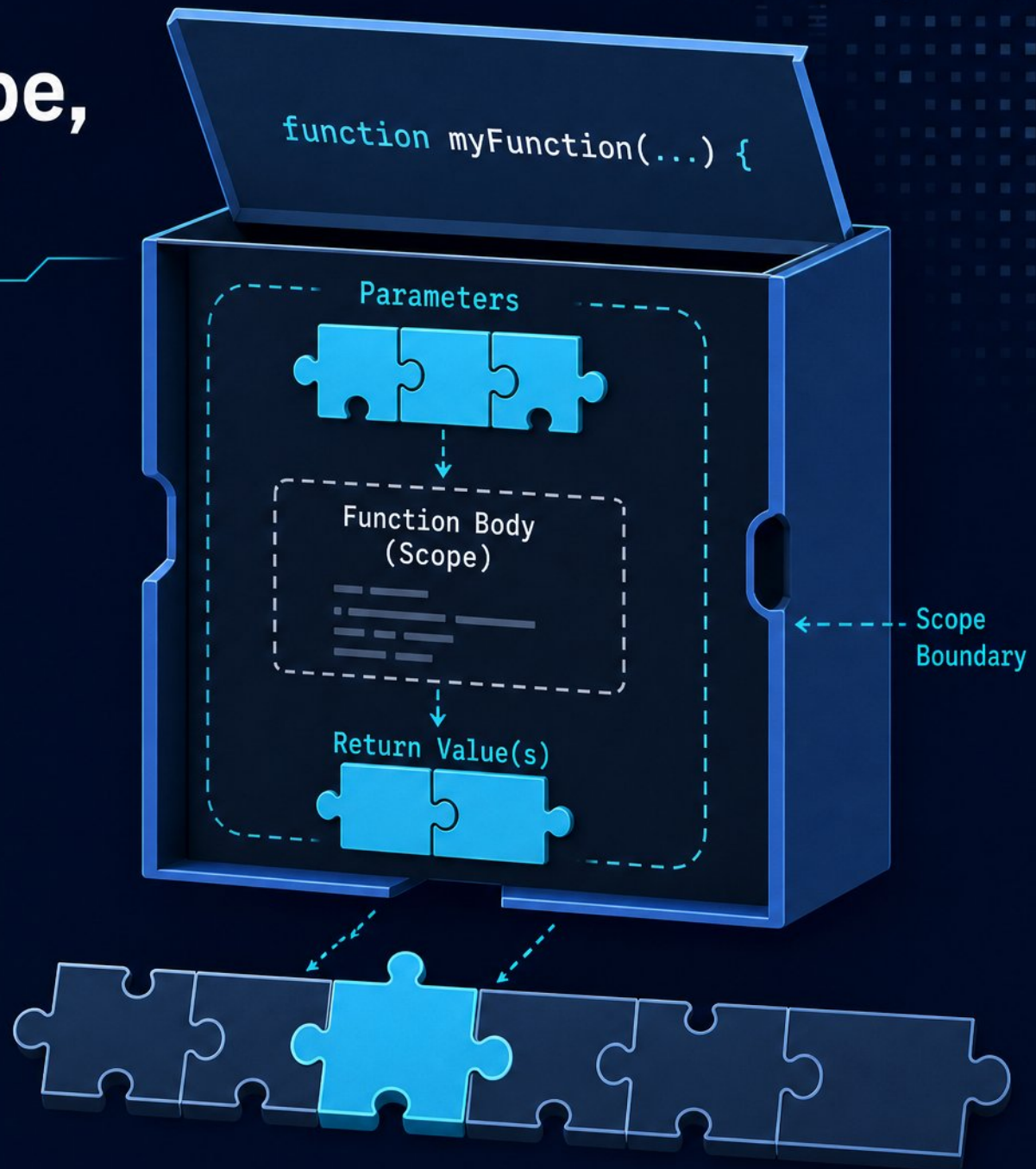
- Refactor repeated code into reusable functions



- Break problems into small testable units



- Practice: primality check, validators, game logic



Debugging and Testing Practice

-  - Read tracebacks to spot error types quickly
-  - Use prints and assertions to isolate bugs
-  - Write small test cases for your solutions
-  - Define expected behavior before coding
-  - Avoid off-by-one, type mismatch, and mutable default pitfalls

```
Traceback (most recent call last):  
  File "solution.py", line 23, in <module>  
    result = divide(a, b)  
  File "solution.py", line 9, in divide  
    return a / b  
ZeroDivisionError: division by zero
```

Prints

```
print("x:", x)  
print("y:", y)  
print("result:", res)
```

Assertions

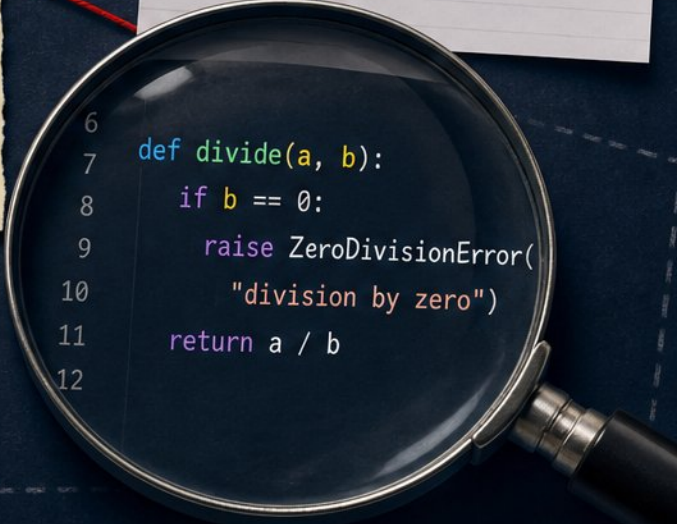
```
assert b != 0  
assert isinstance(x, int)  
assert result >= 0
```

Test Cases

```
def test_divide_positive():  
    assert divide(10, 2) == 5  
def test_divide_negative():  
    assert divide(-9, 3) == -3  
def test_divide_by_zero():  
    try:  
        divide(5, 0)  
    except ZeroDivisionError:  
        pass
```

Expected Behavior

- divide(a, b) returns a / b when b != 0
- raises ZeroDivisionError when b == 0



```
6  
7 def divide(a, b):  
8     if b == 0:  
9         raise ZeroDivisionError(  
10             "division by zero")  
11     return a / b  
12
```

Real-World Mini-Projects: From Scripts to Solutions

- Apply core skills in practical projects
- Examples: number guessing game, word counter, contact book
- Break projects into milestones and subtasks
- Organize code for readability in multi-function programs
- Iterate: add difficulty levels, persistence, better UX



Case Study:

Building a Number Guessing Game

- Random number generation, loops, and input validation
- Difficulty levels adjust range and attempts
- Scoring system tracks and saves high scores
- Refactor into reusable functions
- Extend with JSON persistence or GUI

```
$ python guess_game.py
Welcome to the Number Guessing Game!
Select difficulty:
1) Easy    2) Medium    3) Hard
Enter choice: 2
I have selected a number.
Try to guess it!

Enter your guess: 50
Too high!

Enter your guess: 25
Too low!

Enter your guess: 37
Correct! You guessed it!

Thanks for playing!
$ █
```

Random number generation, loops, and input validation

Difficulty levels adjust range and attempts

Scoring system tracks and saves high scores

Refactor into reusable functions

Extend with JSON persistence or GUI

Curated Problem Sets for Continued Learning

- 20–30 minutes daily, one topic per week
- Warm-up and syntax drills for daily practice
- Logic and algorithm challenges by difficulty
- Project-style exercises mimic real development tasks
- Select problems matching your current level and goals

	EASY	MEDIUM	HARD
DATA STRUCTURES			
ALGORITHMS			
DEVELOPMENT TASKS			

Practice Platforms and Their Strengths



- Exercism: free, 82 tracks, mentor feedback



- Codewars: gamified katas for language fluency



- LeetCode & HackerRank: interview prep with algorithms



- PYNative & W3Schools: structured, topic-based exercises



- Match platform to goal: fluency, interviews, or projects

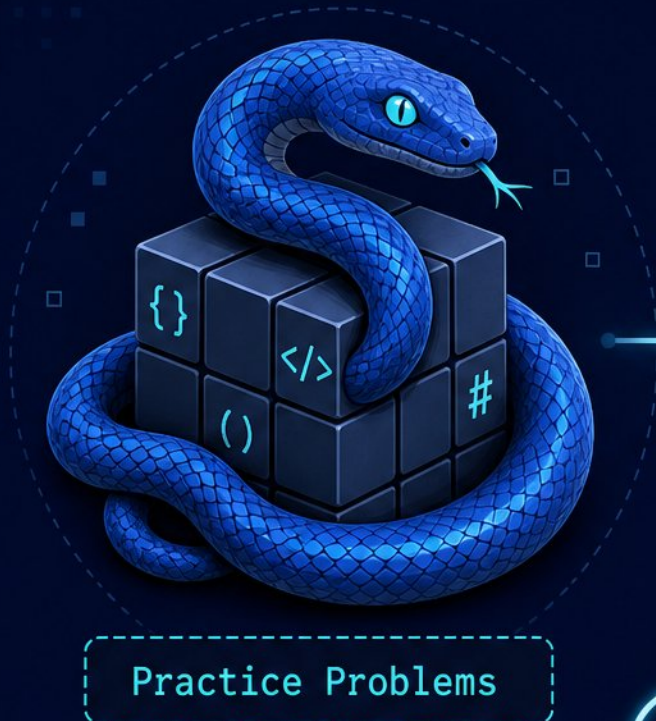
```
{  
  _____  
  _____  
  _____  
  _____  
  _____  
}
```

Building a Personal Practice Routine

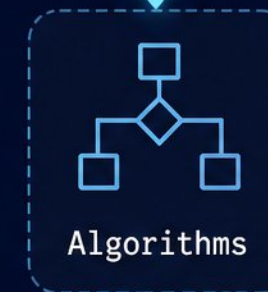
- Mix targeted drills, combined challenges, and small projects
- 8-week plan: foundations → combined concepts → projects
- Track progress with streaks, XP, or a personal problem log
- Seek code review and mentor feedback for idiomatic style
- Transition from exercises to original projects and frameworks



Next Steps: From Practice Problems to Real Projects



- Recognize readiness: solve integrated challenges confidently
- Choose a first project aligned with personal interests
- Start with Git and structured project organization
- Contribute to open source and community challenges
- Paths: data analysis, web development, automation, algorithms



PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/09da34c3/public