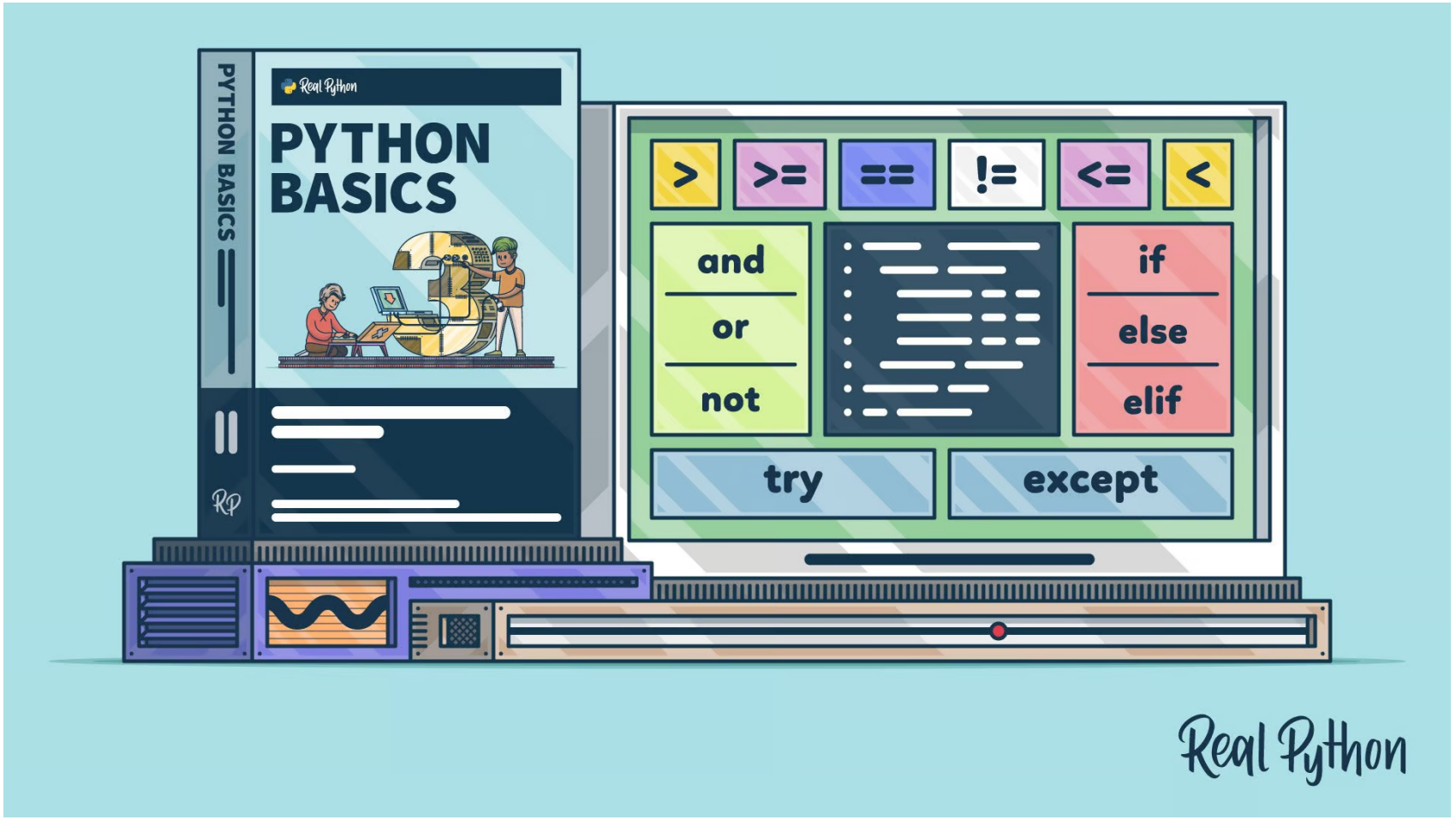
Python Basics Exercises: Conditional Logic and Control Flow

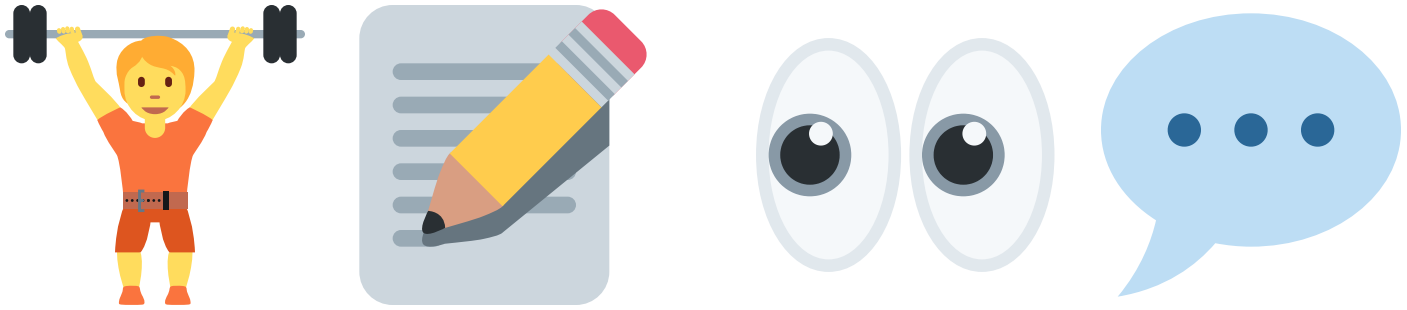


Real Python Exercises Course

Real Python Exercises Course

Real Python Exercises Course

Real Python Exercises Course



Real Python Exercises Course

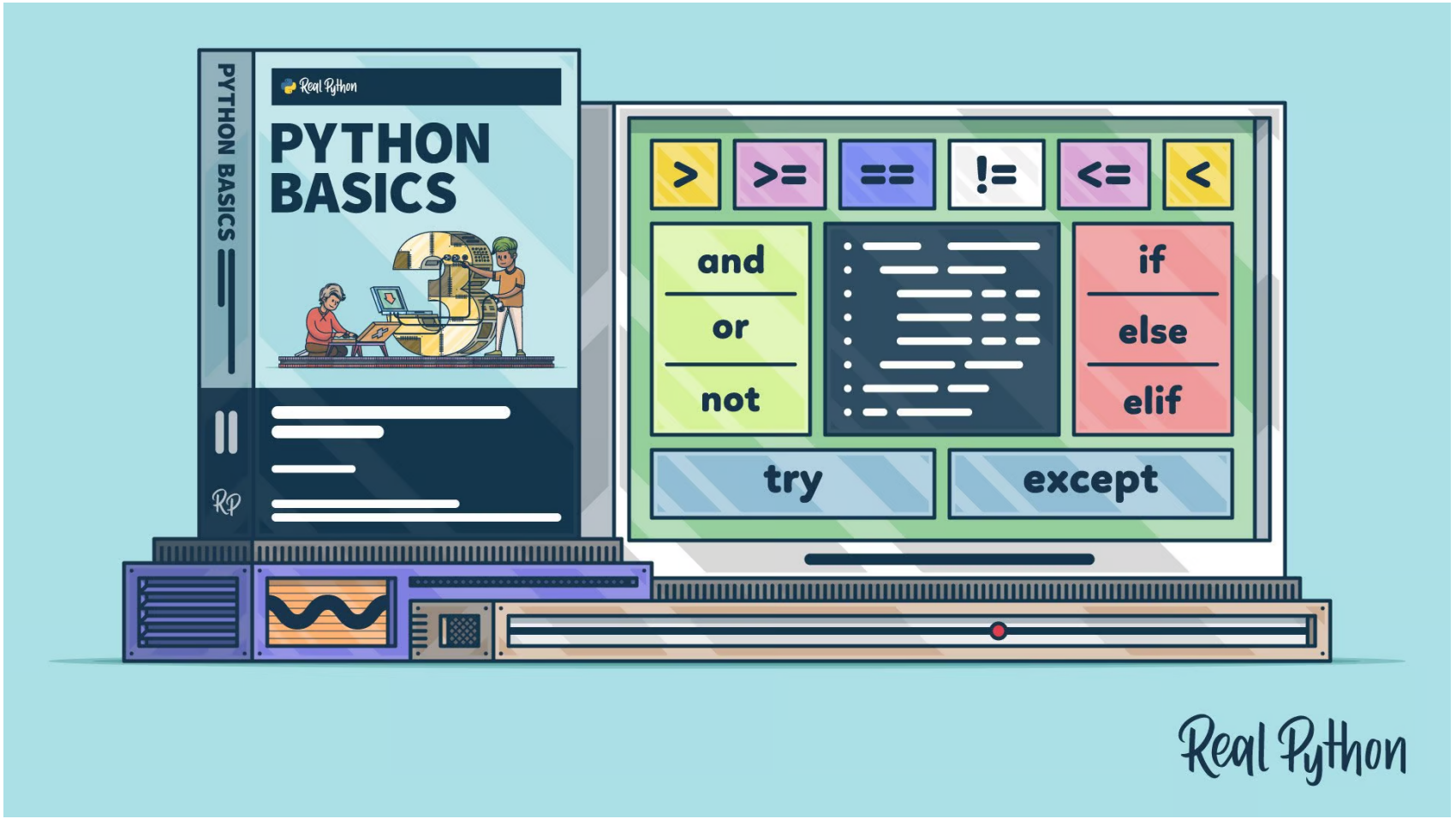
The three steps for each task:

1. 👀 Learn about the exercise
2. 👨‍💻 Code your solution
3. ↔️ Compare your solution

Python Basics Exercises: Conditional Logic and Control Flow

1. Review Exercises
2. Challenges

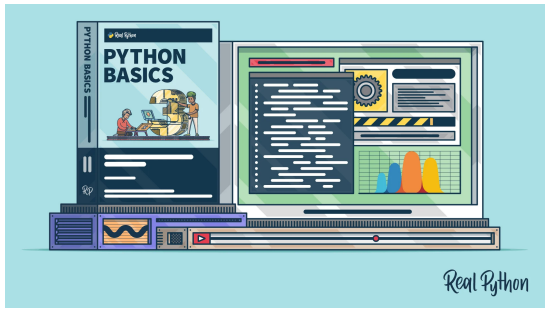
Background - Python Basics: Conditional Logic and Control Flow



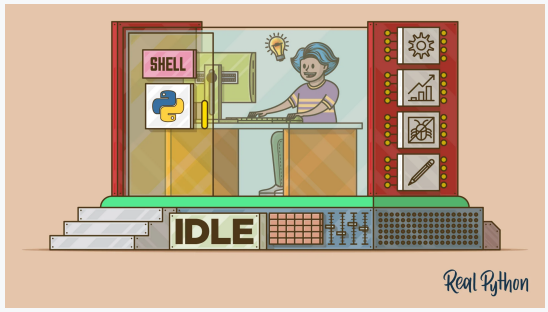
Background - Python Basics: Conditional Logic and Control Flow

- Boolean comparators: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Logical operators: `and`, `or`, `not`
- Conditional logic: `if...elif...else`
- Exception handling: `try...except`
- Loops: `for`, `while`
- Control flow statements: `break`, `continue`

Background - Using IDLE

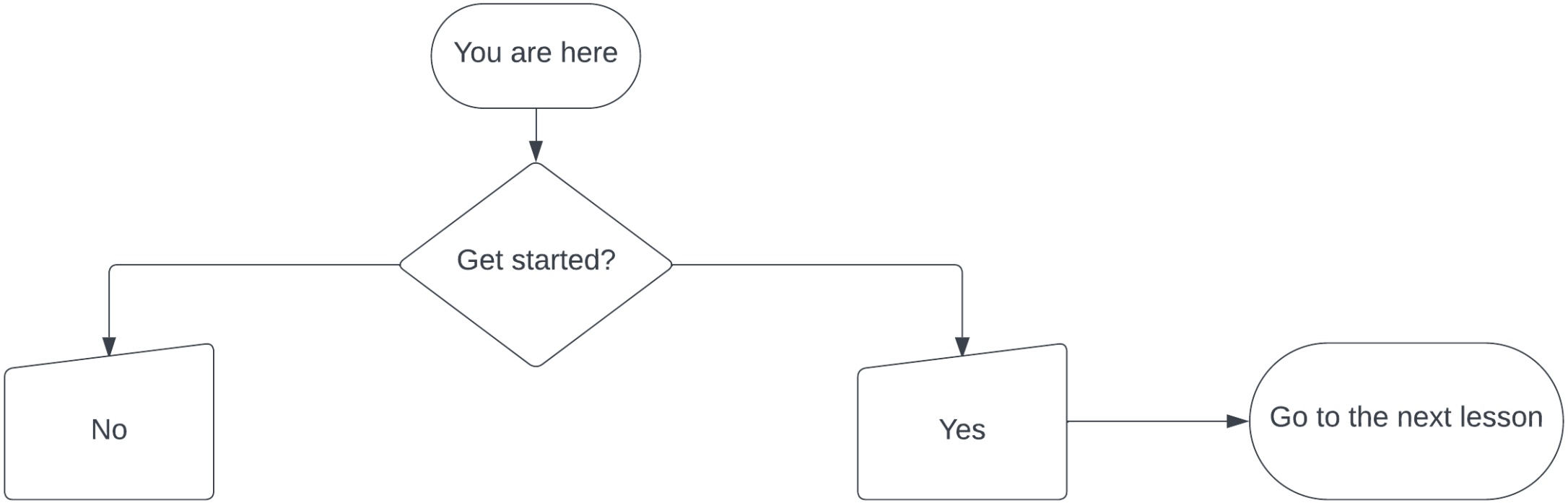


Python Basics: Setting Up Python



Getting Started With Python IDLE

Decision Time: Ready to Get Started?



🤔 Compare Values - Exercise 1

For each of the following conditional expressions, guess whether they evaluate to `True` or `False`. Then type them into the interactive window to check your answers:

- `1 <= 1`
- `1 != 1`
- `1 != 2`
- `"good" != "bad"`
- `"good" != "Good"`
- `123 == "123"`

💡 Compare Values - Exercise 1

For each of the following conditional expressions, guess whether they evaluate to ✅ `True` or ❌ `False`. Then type them into the interactive window to check your answers:

- ✅ `1 <= 1`
- ❌ `1 != 1`
- ✅ `1 != 2`
- ✅ `"good" != "bad"`
- ✅ `"good" != "Good"`
- ❌ `123 == "123"`

🤔 Compare Values - Exercise 2

For each of the following expressions, fill in the blank (indicated by `__`) with an appropriate Boolean comparator so that the expression evaluates to `True`:

- `3 __ 4`
- `10 __ 5`
- `"jack" __ "jill"`
- `42 __ "42"`

💡 Compare Values - Exercise 2

For each of the following expressions, fill in the blank (indicated by `__`) with an appropriate Boolean comparator so that the expression evaluates to ✅ `True`:

- ✅ `3 < 4`
- ✅ `10 > 5`
- ✅ `"jack" < "jill"`
- ✅ `42 != "42"`

🤔 Add Logic - Exercise 1

Figure out what the result will be (`True` or `False`) when evaluating the following expressions, then type them into the interactive window to check your answers:

- `(1 <= 1) and (1 != 1)`
- `not (1 != 2)`
- `("good" != "bad") or False`
- `("good" != "Good") and not (1 == 1)`

💡 Add Logic - Exercise 1

Figure out what the result will be (✅ `True` or ❌ `False`) when evaluating the following expressions, then type them into the interactive window to check your answers:

- ❌ `(1 <= 1) and (1 != 1)`
- ❌ `not (1 != 2)`
- ✅ `("good" != "bad") or False`
- ❌ `("good" != "Good") and not (1 == 1)`

🤔 Add Logic - Exercise 2

Add parentheses where necessary so that each of the following expressions evaluates to `True`:

- `False == not True`
- `True and False == True and False`
- `not True and "A" == "B"`

💡 Add Logic - Exercise 2

Add parentheses where necessary so that each of the following expressions evaluates to ✅ `True`:

- ✅ `False == (not True)`
- ✅ `(True and False) == (True and False)`
- ✅ `not (True and "A" == "B")`

🤔 Control the Flow of Your Program

Write a program that prompts the user to enter a word using the `input()` function and compares the length of the word to the number five. The program should display one of the following outputs, depending on the length of the user's input:

- `"Your input is less than 5 characters long"`
- `"Your input is greater than 5 characters long"`
- `"Your input is 5 characters long"`

💡 Control the Flow of Your Program

```
my_input = input("Type something: ")

if len(my_input) < 5:
    print("Your input is less than 5 characters long.")
elif len(my_input) > 5:
    print("Your input is greater than 5 characters long.")
else:
    print("Your input is 5 characters long.")
```

🤔 Break Out of the Pattern - Exercise 1

Using `break`, write a program that repeatedly asks the user for some input and quits only if the user enters `"q"` or `"Q"`.

Break Out of the Pattern - Exercise 1

```
while True:
    user_input = input('Type "q" or "Q" to quit: ')
    if user_input.upper() == "Q":
        break
```

🤔 Break Out of the Pattern - Exercise 2

Using `continue`, write a program that loops over the numbers `1` to `50` and prints all numbers that are not multiples of `3`.

Break Out of the Pattern - Exercise 2

```
for i in range(1, 51):
    if i % 3 == 0:
        continue
    print(i)
```

🤔 Recover from Errors

Write a program that repeatedly asks the user to input an integer. If the user enters something other than an integer, then the program should catch the `ValueError` and display the message `"Try again."` Once the user enters an integer, the program should display the number back to the user and end without crashing.

Recover from Errors

```
while True:
    try:
        my_input = input("Type an integer: ")
        print(int(my_input))
        break
    except ValueError:
        print("try again")
```

🤔 Simulate Events

Write a function called `roll()` that uses `randint()` to simulate rolling a fair die by returning a random integer between `1` and `6`.

💡 Simulate Events

```
from random import randint

def roll():
    """Return random integer between 1 and 6"""
    return randint(1, 6)
```

Review Exercises: Recap

- Boolean comparators: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Logical operators: `and`, `or`, `not`
- Conditional logic: `if...elif...else`
- Exception handling: `try...except`
- Loops: `for`, `while`
- Control flow statements: `break`, `continue`

Tips

- Use code comments to help you get organized
- Break exercises into smaller tasks
- Use descriptive variable names
- Test repeatedly to see whether the code does what you expect it to do

Tips

- Use code comments to help you get organized
- Break exercises into smaller tasks
- Use descriptive variable names
- Test repeatedly to see whether the code does what you expect it to do

Next Up: Challenges!

🤔 Challenge: Find the Factors of a Number

A factor of a positive integer `n` is any positive integer less than or equal to `n` that divides `n` with no remainder.

For example, `3` is a factor of `12` because `12` divided by `3` is `4` with no remainder. However, `5` is not a factor of `12` because `5` goes into `12` twice with a remainder of `2`.

🤔 Challenge: Find the Factors of a Number

Write a program called `factors.py` that asks the user to input a positive integer and then prints out the factors of that number. Here's a sample run of the program with output:

```
Enter a positive integer: 12
1 is a factor of 12
2 is a factor of 12
3 is a factor of 12
4 is a factor of 12
6 is a factor of 12
12 is a factor of 12
```

💡 Challenge: Find the Factors of a Number

```
num = int(input("Enter a positive integer: "))
for divisor in range(1, num + 1):
    if num % divisor == 0:
        print(f"{divisor} is a factor of {num}")
```

🤔 Challenge: Build a Text-Based RPG Game

Write a text-based role-playing game (RPG) with the following characteristics:

- There's one player and one monster
- The player starts with `100` health, the monster with `150` health (it's big!)
- The player can choose to attack, heal, or run away.
- If they attack, they deal between `10` - `15` damage to the monster.
- If they heal, they regain `30` health, up to a maximum of `100`.
- If they run away, the game ends.
- After the player's turn, if the monster is still alive, it deals `15` - `20` damage to the player.

The game continues until either the player or the monster's health reaches `0`, or the player runs away.

🤔 Challenge: Build a Text-Based RPG Game

Stretch Goals

- Prevent the player from exiting the game by pressing `Ctrl` + `C`.
- Introduce double-damage critical hits that happen when an attack value is a factor of `3`.
- Use emojis or string formatting syntax to make the game output more fun.

✅ Challenge: Build a Text-Based RPG

Write a text-based role-playing game (RPG) with the following characteristics:

- There's one player and one monster
- The player starts with `100` health, the monster with `150` health (it's big!)
- The player can choose to attack, heal, or run away.
- If they attack, they deal between `10` - `15` damage to the monster.
- If they heal, they regain `30` health, up to a maximum of `100`.
- If they run away, the game ends.
- After the player's turn, if the monster is still alive, it deals `15` - `20` damage to the player.

The game continues until either the player or the monster's health reaches `0`, or the player runs away.

✅ Challenge: Build a Text-Based RPG

✅ Stretch Goals

- Prevent the player from exiting the game by pressing `Ctrl` + `C`.
- Introduce double-damage critical hits that happen when an attack value is a factor of `3`.
- Use emojis or string formatting syntax to make the game output more fun.

Summary and Additional Resources

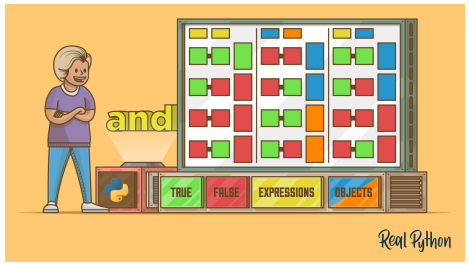
In this course, you **practiced** using:

- Boolean comparators: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Logical operators: `and`, `or`, `not`
- Conditional logic: `if...elif...else`
- Exception handling: `try...except`
- Loops: `for`, `while`
- Control flow statements: `break`, `continue`

Tips

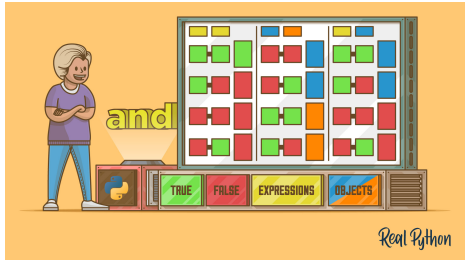
- Use code comments to help you get organized
- Break exercises into smaller tasks
- Use descriptive variable names
- Test repeatedly to see whether the code does what you expect it to do

Additional Resources

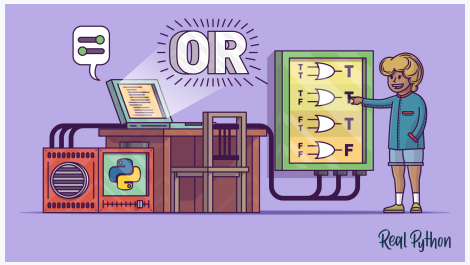


Conditional Statements in Python

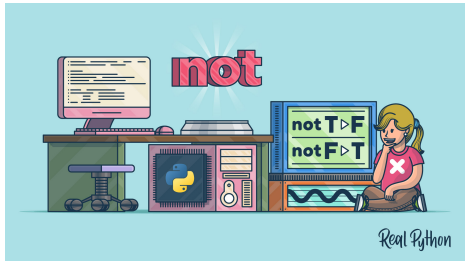
Additional Resources



Using the “and” Boolean Operator in Python



Using the “or” Boolean Operator in Python



Using the “not” Boolean Operator in Python

Additional Resources

Python “for” Loops (Definite Iteration)

Python “while” Loops (Indefinite Iteration)

Python Exceptions: An Introduction

Congratulations and Thanks!