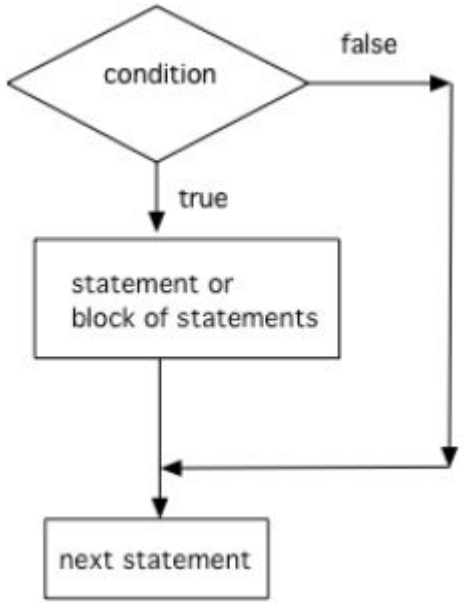
Starting up at 5:35

(Re)-Introduction to Data Science & Control Flow

Agenda - Schedule

1. Warm Up
2. Canvas & Data Science Review
3. Docstrings in Functions
4. Control Flow & Truthiness
5. Nested if Statements
6. VSCode Lab



Control flow diagram
https://runestone.academy/ns/books/published/csawesome/Unit3-If-Statements/topic-3-2-ifs.html

Agenda - Goals

- Get re-introduced to data science
- Understand how to document your functions
- Review basic conditionals and implement nested conditionals in Python

Warm-Up

```
def list_process(data: list, n: int) -> list:
    averages = []
    if len(data) <= 1 or n == 1:
        return averages

    window = data[:n]
    roll_average = sum(window ) / n
    averages.append(roll_average)

    return averages

print(list_process([], 1))
print(list_process([2, 1, 3, 4, 6, 5], 1))
print(list_process([2, 1, 3, 4, 6, 5], 3))
```

Evaluate this chunk of code. Work together to figure out what will occur when we run this code. Try not to run this code in your editor, and instead, evaluate what occurs to the data at each step.

You are working as a data analyst at an online shopping platform.

You are tasked with performing descriptive customer segmentation by evaluating one dimension: age.

Which visualization would be most appropriate to express the distribution of the ages of your customers? Hint: *are we performing predictive or descriptive analytics? Do we have one or multiple dimensions? Are our dimensions quantitative or categorical?*

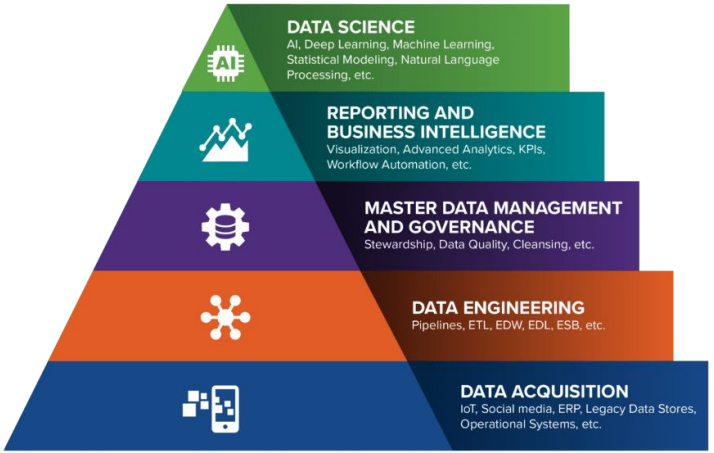
Join your pod groups and evaluate this statement. Work together to figure out what the appropriate visualization would be.

Fellowship Purpose

Course Goals - Abstract

In the shallow sense of things, this fellowship aims to teach you different skills that **build off of each other**. In the abstract, the goals entail:

Phase 1: *Learn how to engineer, pipeline, and collect*
Phase 2: *Learn how to analyze, predict, model, and innovate*
Phase 3: *Learn how to manage, collaborate, visualize, and inform*



Course Goals

*“By the end of this fellowship, you will have a **comprehensive suite** of projects and a thorough foundation of data-science knowledge that will allow you to **succeed and grow** in any job that uses a **contemporary tech stack** to answer **complex data-driven questions**.”*

Some Tips for the Upcoming Weeks

- ***Take things slowly,*** *and try not to worry about material we didn't cover yet. Focus on the now.*
- *We rarely understand things at first glance.* ***Revisit topics often.***
- *If you are feeling challenged, that is a good sign!* ***Stay with that discomfort.***
- *Keep your* ***personal goals and intellectual curiosities*** *in mind as you complete this fellowship. Listen to podcasts, write, watch sci-fi, etc.*
- ***Make time to step away from the computer.***
- ***Communicate early and often.*** *It's easy to let things slip in a remote environment*

Fellowship Structure

Phase 3 will arguably be the most important phase! You will work in your pods to create a comprehensive, end-to-end data science project.

Schedule Overview

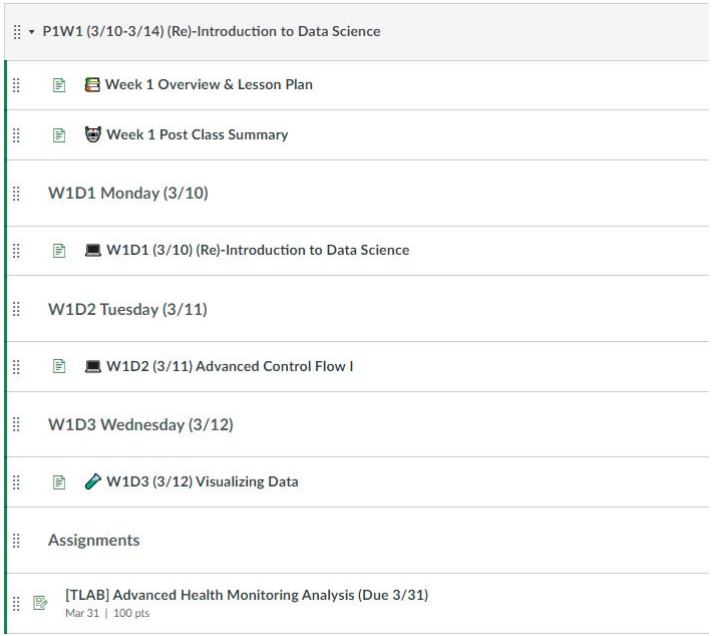
Phase	Dates	Assignments	Quizzes	Focus
1	03/10/2025 - 05/23/2025	1 End of Phase Project 3 TLABS	10 Quizzes	Python, pandas, Computational Thinking, Introductory Statistics, Web-Scraping, API, SQL
2	06/09/2025 - 08/29/2025	1 End of Phase Project 3 TLABS	10 Quizzes	Supervised & Unsupervised machine learning, SQL, Generative AI
3	09/15/2025 - 12/05/2025	1 Capstone Project		Group Capstone

Class Structure

Each week of class will entail the following content:

1. Week Overview & Lesson Plan
2. Post-class Summary
3. Class Content
4. A long-term **assignment** or short-term **quiz**

All together, we recommend that you spend 10-20 hours/week outside of class working on material for your success.



Weekly Overview

This page will inform you about incoming content/assignments.

In addition, it will also list all resources/videos we would like for you to watch to “get ahead” of class material.

When the fellowship starts, this page will also **entail a light 5-question quiz to prepare you for the week, due before Monday’s class.**

No quiz this week since its week 1.

Post-Class Summary

We will also post a post-class summary to wrap the week up.

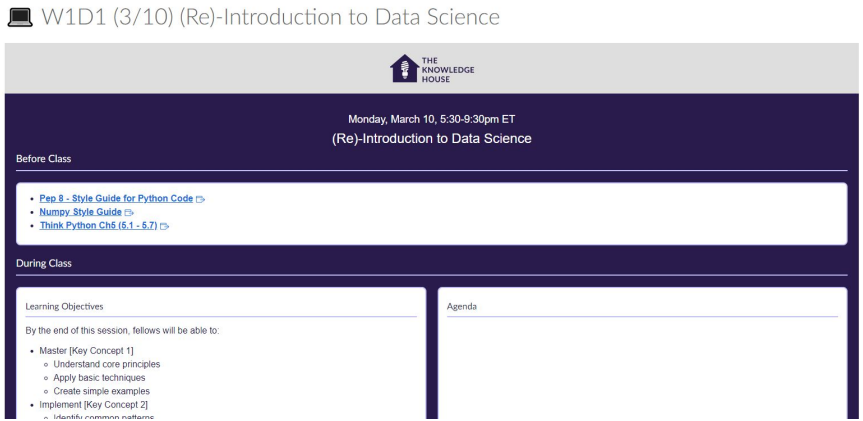
This will include:

- An AI “Podcast” summary on the respective weeks content
- Extra resources for those looking for **additional challenges** (highly highly recommended)
- **Incoming assignments**

Class Content

This page will inform you about the daily activity/resources that you need to follow along with class material.
This entails:

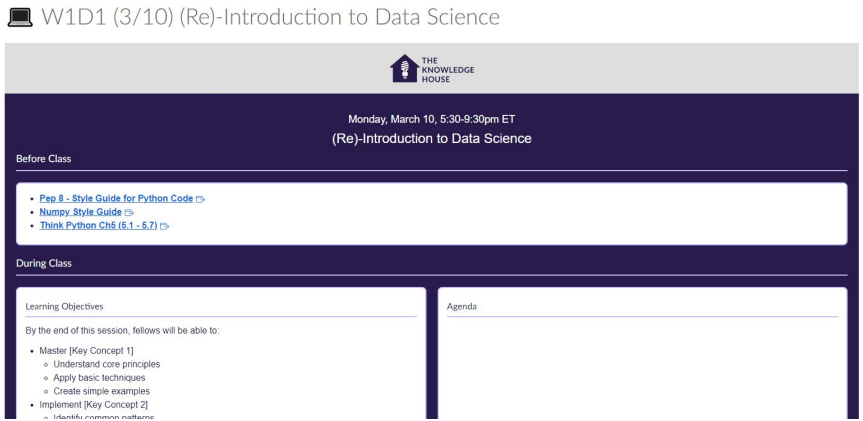
- ***Slides***
- ***Readings***
- ***Videos***
- ***Exercises***



Stats Lab Content

Each Wednesday, we will switch gears and solely **focus on a statistics module** to complement your programming knowledge.

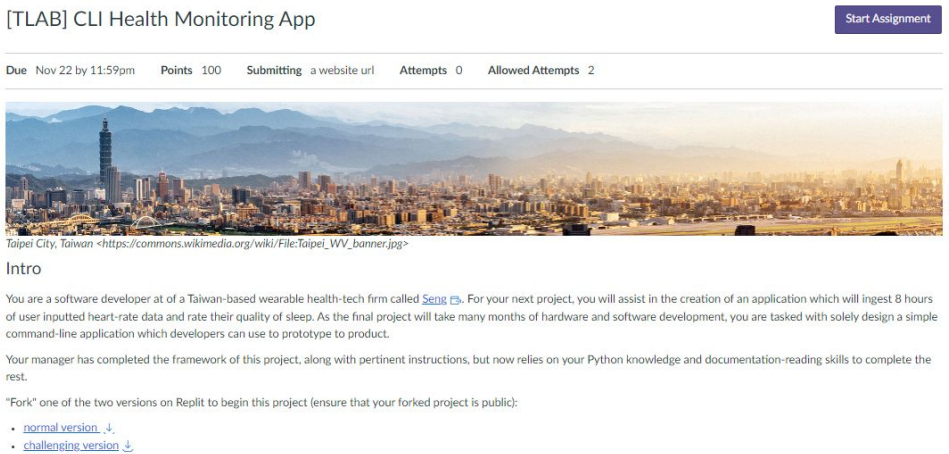
This will be run like a typical class day.



TLAB/Quiz

Once every 3 weeks we will post a **long-term TLAB** which you will work on outside of class, as well as during the Thursday review sessions.

Be sure to start early.



[TLAB] CLI Health Monitoring App

Start Assignment

Due Nov 22 by 11:59pm **Points** 100 **Submitting** a website url **Attempts** 0 **Allowed Attempts** 2

Taipei City, Taiwan <https://commons.wikimedia.org/wiki/File:Taipei_WV_banner.jpg>

Intro

You are a software developer at of a Taiwan-based wearable health-tech firm called Seng. For your next project, you will assist in the creation of an application which will ingest 8 hours of user inputted heart-rate data and rate their quality of sleep. As the final project will take many months of hardware and software development, you are tasked with solely design a simple command-line application which developers can use to prototype to product.

Your manager has completed the framework of this project, along with pertinent instructions, but now relies on your Python knowledge and documentation-reading skills to complete the rest.

"Fork" one of the two versions on Replit to begin this project (ensure that your forked project is public):

- normal version
- challenging version

I am a fairly long-winded person, so be sure to save questions for the Zoom chat/1:1 office hours.

Class Structure

Unless the syllabus states otherwise, our days will entail the following schedule:

Monday, Tuesday, Wednesday

5:30 - 7:30 PM: *Lecture & code-along*

7:30 - 8:00 PM: *Break*

8:00 - 9:30 PM: *Lab session*

Thursday

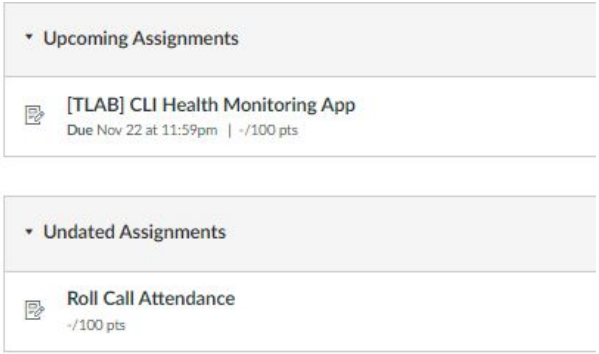
5:30 - 7:30 PM: *Open review session*

Canvas/Assignment Structure

Assignment Structure

This fellowship will follow a predictable format for phase 1 and phase 2.

- ***3 TLAB's (3 week due date)***
  - *30% of grade*
- ***10 quizzes (1 week due date)***
  - *20% of grade (lowest dropped)*
- ***1 End of Phase Project (4 week due date)***
  - *50% of grade*



Occasionally, you will have to look to outside resources to completes these assignments!

Assignment Structure - Resubmissions

Each TLAB **can be submitted twice** before **the due date** (highest grade will be kept).

If you submit **within 1 week of the due date** you will receive feedback .

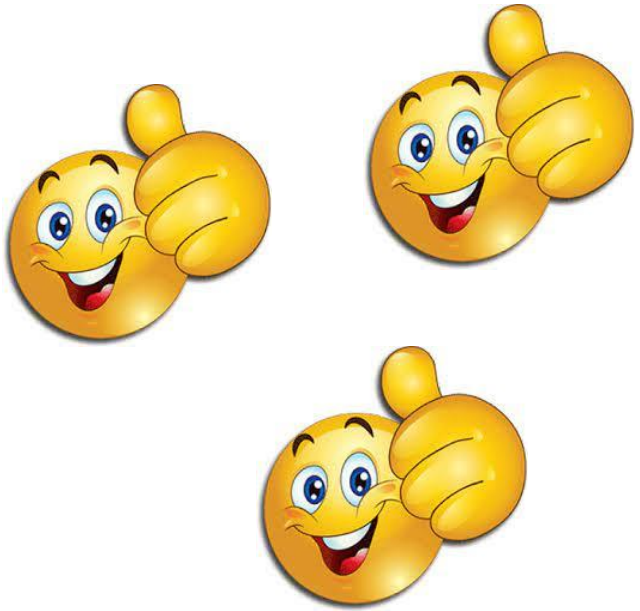
Feedback is one of the most valuable things you can receive as a technologist, use it!!

Assignment Structure - Late Submissions

During each phase you get 3 “tokens” that **you can use to submit an assignment 48 hours late.**

However you cannot “stack” tokens for one assignment.

Any accommodation beyond these 3 needs to be requested to the Instructor & Student Success specialist.



(Re)-Introduction to Data Science

Those who show up make the decisions.

(Re)-Introduction to Data Science

We've gone over the **history**, **prospects**, and **importance** of data science.

Let's close out this discussion by **talking about the ever-creeping future of data science** and why your presence in this **fellowship matters**.

Once again, we are not historians, political scientists, nor economists, but we can talk about atomic realities which reveal trends.

While these are harsh realities to acknowledge, they also prepare us for which challenges we need to surmount.

(Re)-Introduction to Data Science : Market

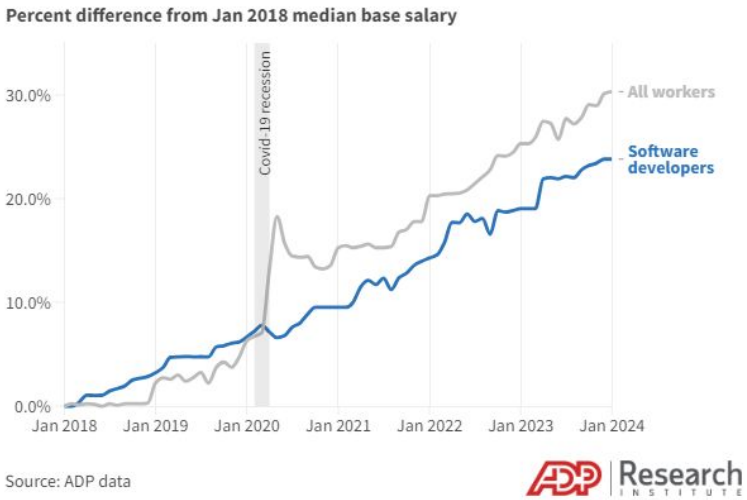
Currently we are in what we call a **buyers market** (*employers favor*) whereas most of the previous decade favored the **sellers market** (*employees favor*).

This means:

- **Lower starting salaries**, with less room to negotiate. **Stagnation**.
- Starkly **more competition** for entry level roles.
- **Less priority** for companies to attract talent through **benefits**.
- Also less priority for employers to **match employee culture**.

Software developer salaries grew slower than average from 2018 to 2024

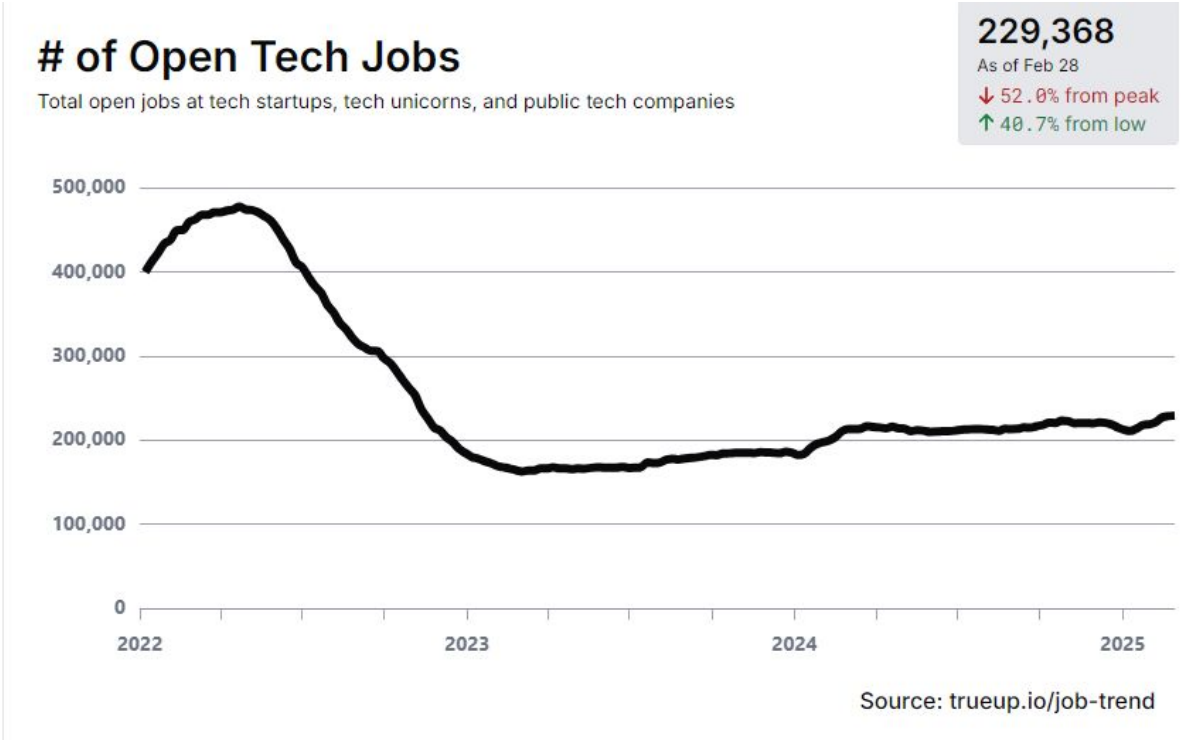
Before the pandemic, software developer salaries grew faster than average. That changed when the Covid-19 recession ended.



Source: ADP data

ADP Research Institute

Using software engineer roles as a proxy, we see that salaries are **growing slower.**



The hiring bubble of tech roles during COVID is gone (no more cheap capital) and open **tech roles reflect a more tepid hiring market.**


TECH

Google CEO Pichai tells employees not to 'equate fun with money' in heated all-hands meeting

PUBLISHED FRI, SEP 23 2022•9:14 AM EDT | UPDATED MON, SEP 26 2022•1:15 PM EDT

Jennifer Elias
@JENN_ELIAS

SHARE


BUSINESS • DONALD TRUMP

The Major U.S. Companies Scaling Back DEI Efforts as Trump Targets Initiatives

8 MINUTE READ

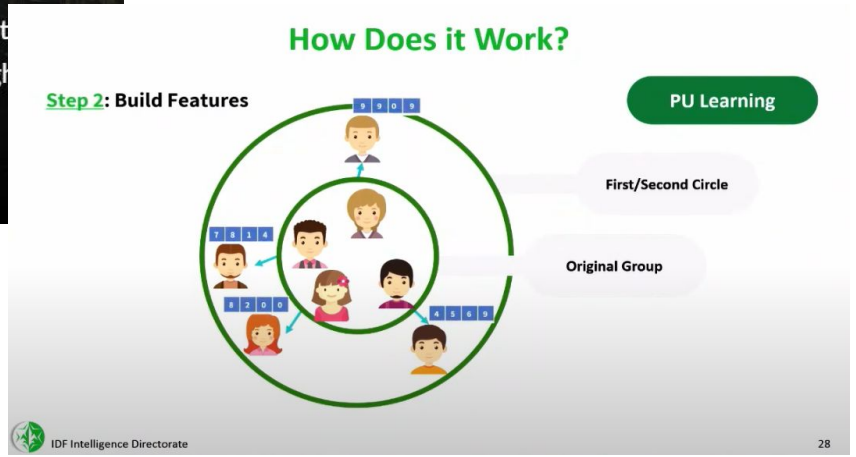
Part of this is political alignment and an assessment of what "America" wants, but companies are **deprioritizing initiatives** to align themselves with employee culture.

*Keep in mind that none of these are inherently **political statements**. I'm operating off of a simple pre-condition: "using algorithms to kill people & create a surveillance state is bad."*

(Re)-Introduction to Data Science : Future Products

In addition to a deprioritization of culture alignment the data science products of tomorrow might be **direct implementations** of the **dystopian sci-fi movies of the past century**.

I don't have the data to prove this yet, and this **might seem alarmist** but check out what's **being actively used and funded**...

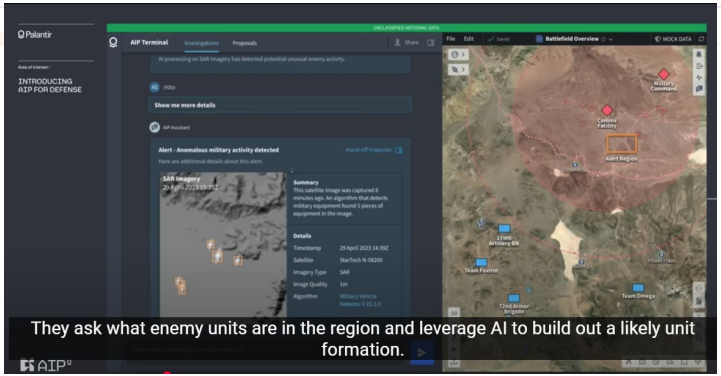


In the lecture, the commander speaks about a new, sophisticated target machine used by the Israeli army that detects “dangerous people” based on their likeness to existing lists of known militants on which it was trained.
https://www.972mag.com/lavender-ai-israeli-army-gaza/

Enough with the selfies: Top Palantir executives roast Silicon Valley titans for creating photo--sharing apps and chat interfaces rather than tackling national problems

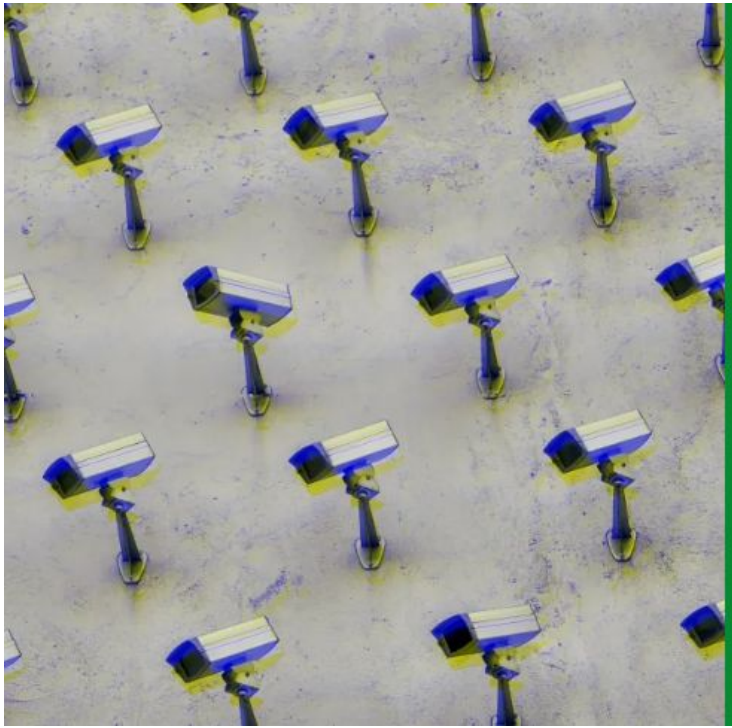
Last Updated: 17 February, 2025 09:36 PM -5 GMT

FOLLOW US SHARE FONT



Understanding the Military AI Ecosystem of Ukraine

"...that, since the country's tech scene is historically a product of the national security state, Silicon Valley executives and engineers should shed their compunctions about working with the military and dedicate themselves to ensuring another century of American hard-power supremacy." https://newrepublic.com/article/191786/alex-karps-war-west-palantir



STARTUPS

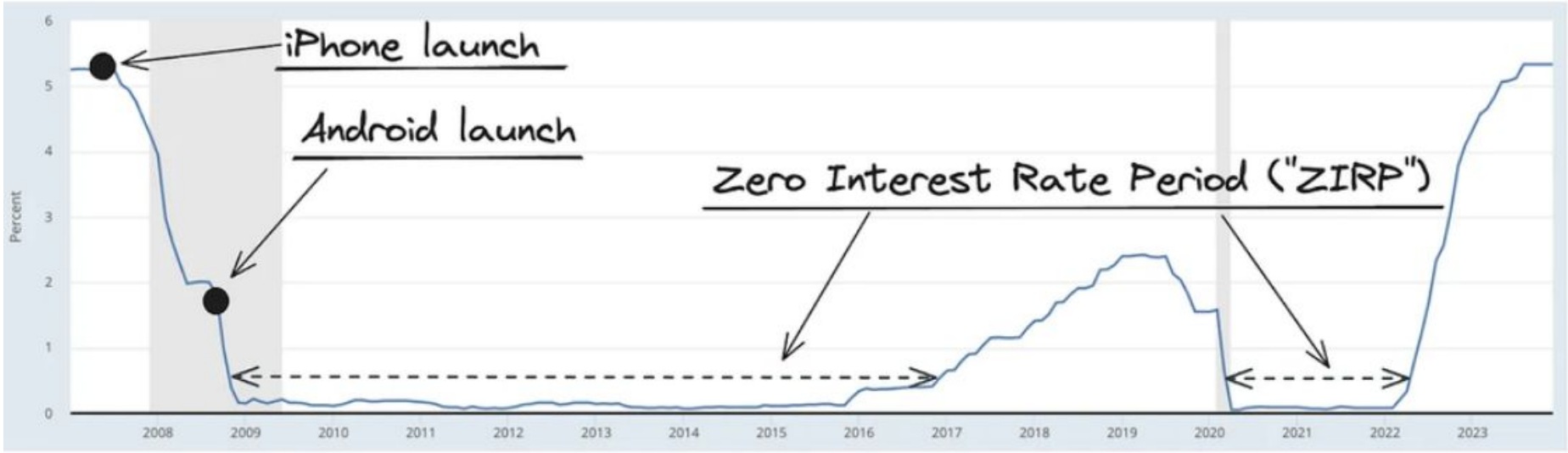
Y Combinator deletes posts after a startup's demo goes viral

Optifye says it's building software to help factory owners know who's working — and who isn't — in "real-time" thanks to AI-powered security cameras it places on assembly lines, according to its YC profile.
https://techcrunch.com/2025/02/25/y-combinator-deletes-posts-after-a-startups-demo-goes-viral/

What all this Means For You

Only time will truly tell what the future holds, but here's what all of this means for you and I:

- This tech crash has **happened before**, and has historically led to a sellers market (*power on employees side*). **You must be patient and your motivation for pursuing this role must go beyond money.**
- If you pursue an attitude of "*The destination matters, not the journey*", you will find yourself working on the equivalent of **SkyNet**.



The market switched sides to where the employees had all the power, and due to another massive round of venture capital injection, money was flowing freely.
https://ghuntley.com/screwed/

DocStrings

Documentation

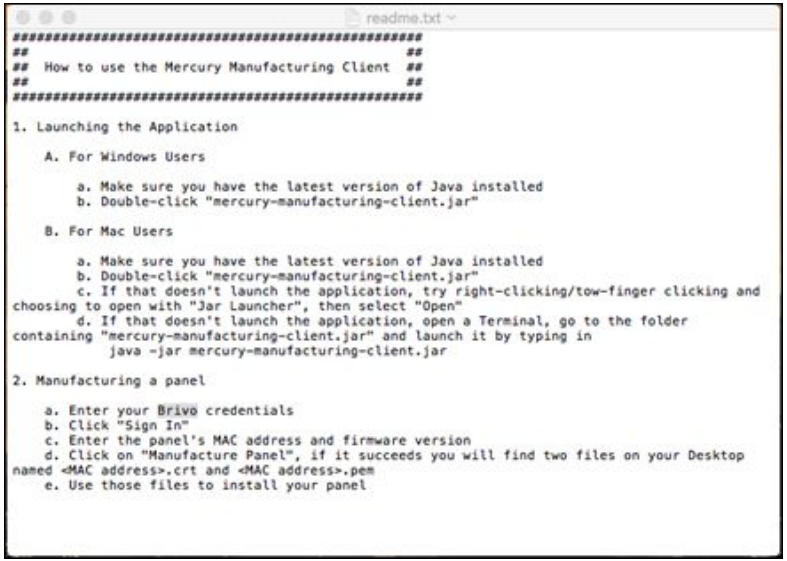
Remember, documentation is the backbone to a maintainable codebase. We will be grading you on your ability to document as well.

Let's revisit the concept of documenting our code.

Documentation comes in two forms when programming:

- ***README file:*** A file that encapsulates the usage and purpose of your project.
- ***Code comments:*** "Comments" that describe discrete pieces of code.

Another aspect to "code comments" is something called the '**docstring**.'



abs(*x*)

Return the absolute value of a number. The argument may be an integer, a floating-point number, or an object implementing `__abs__()`. If the argument is a complex number, its magnitude is returned.

aiter(*async_iterable*)

Return an asynchronous iterator for an asynchronous iterable. Equivalent to calling `x.__aiter__()`.

Note: Unlike `iter()`, `aiter()` has no 2-argument variant.

> *Added in version 3.10.*

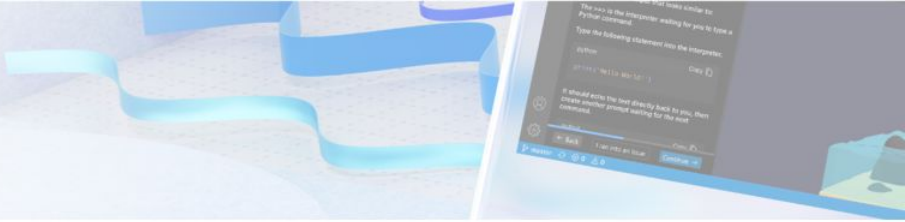
all(*iterable*)

Return `True` if all elements of the *iterable* are true (or if the iterable is empty). Equivalent to:

```
def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True
```

When you consult the documentation of builtin functions, you may notice that there is a **quick blurb describing the function**, as well as motivating examples. This is called the **docstring**, and you will do the same for the functions that you define as well.

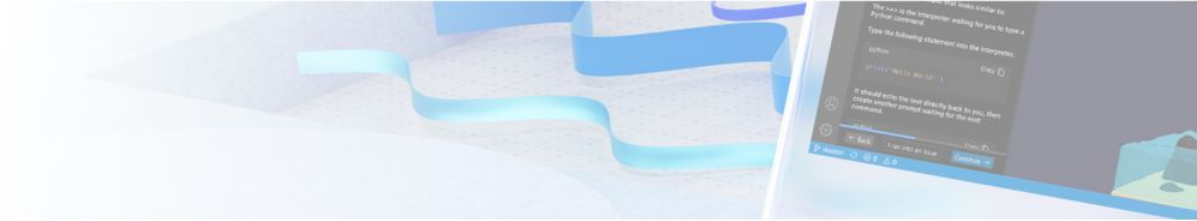
Function Documentation



- Look for six major parts in the documentation:
  - The function's name
  - The syntax of its call
  - Its parameters
  - Its return value
  - Its description
  - Examples of how to use the function
- Sometimes documentation only includes partial information, in which case you must infer the missing details from context.

Name	`len`
Description	The function `len` returns the number of characters in a string (its length).
Syntax	`len(a_string)`
Parameters	`a_string` (String): The string to calculate the length of
Returns	Integer
Example	`len("Hello World")`

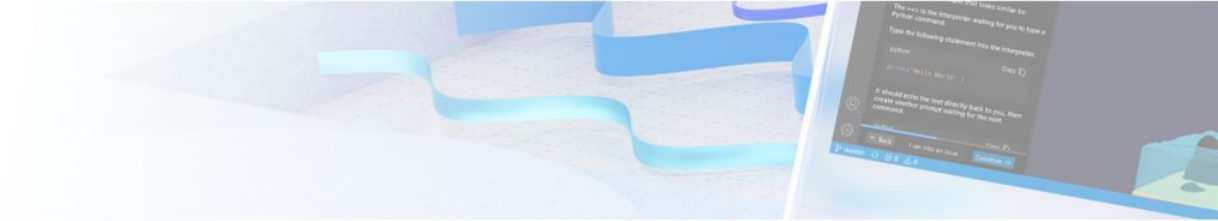
Single-Line Comment



- To make a single-line comment in Python, use the hash symbol (#).
- Everything after the hash symbol on the same line is completely ignored by your program.
- Although programmers often use these comments to document a single line of code, you must not use single-line comments to create docstrings.

```
# This line is ignored
a = 0 # The rest of this line is ignored
b = 0C
```

Multi-Line Comment



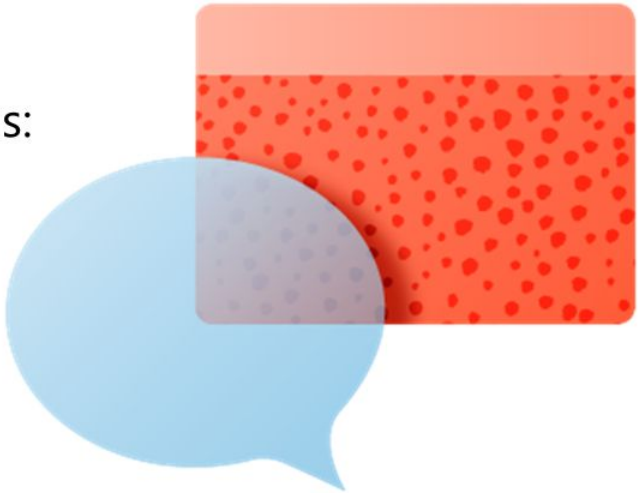
- To write multi-line comments in Python, you create a triple-quoted string (`"""`).
- When Python evaluates this string, the value is unused, so the computer safely ignores it.
- Unlike the single-line comment, the start of a multi-line comment must respect indentation rules as shown in the example.
- Because this triple-quote string counts as the body of the function, you no longer need the `pass` statement.
- You can't have a multi-line string value on a line with any other code.

```
def my_function(x:int, y:int)->int:
    """
    The start of this string needed to be
    indented, otherwise you would have a syntax
    error!
```

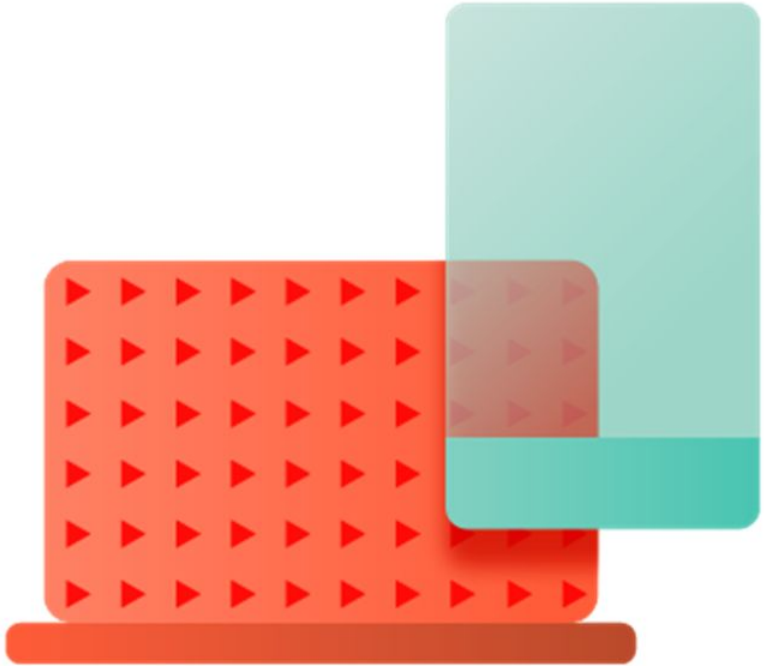
How Much to Comment?

- There's no correct answer for how much to comment — some people believe you need to document every line, while others think you should only document functions and programs.

- Find your own balance between the following options:
  - **High**: Every line of code
  - **Medium**: Only confusing or large chunks of code
  - **Low**: Only functions

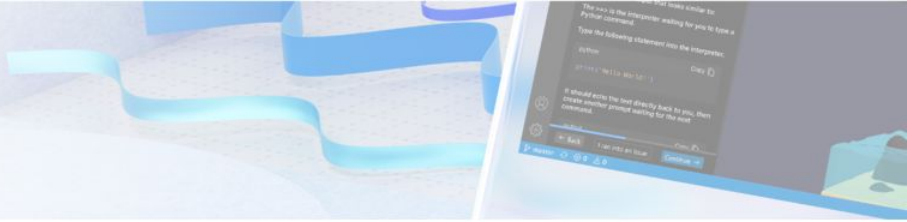


Documenting Functions



- At a minimum, you should document all your functions.
- Use a triple-quoted string as the first line in the function's definition, and then describe what the function does.
- Also explain what the parameters and returned value mean.

Documenting Functions

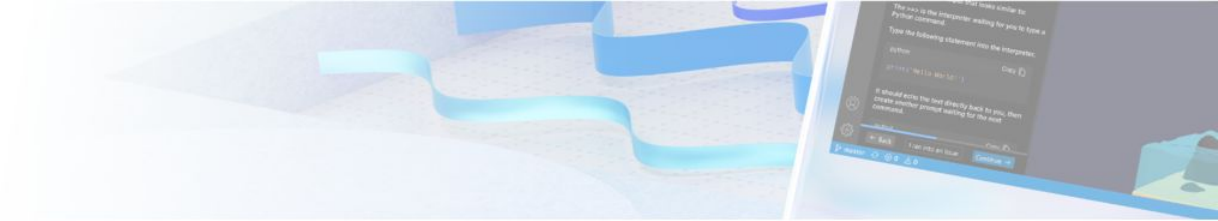


1. Leave a blank line, and then type `Args:`. Indent the next line, and then type the name of the first parameter followed by its type in parentheses.
2. On the same line, write a colon and then describe the parameter.
3. If you need to continue onto the next line, make sure it's indented past the previous line.
4. Repeat for all parameters of your function.
5. Finally, write `Returns:`, and then on the next line, write the name of the return type followed by a colon and a description of the returned value.

```
def fix_capitalization(title: str)->str:
    '''
    This function capitalizes the first letter of each
    word in the title, correctly ignoring prepositions.

    Args:
        title (str): A book or movie title
            that we want to fix.
    Returns:
        str: The corrected title
    '''
```

Missing Docstrings

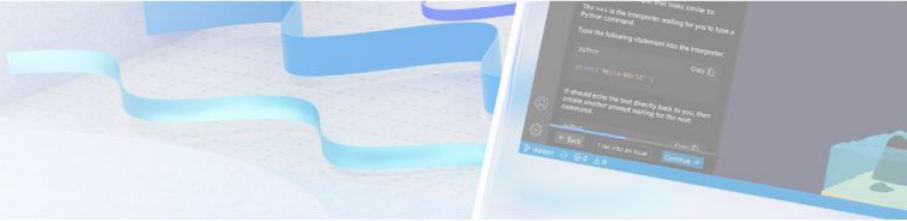


- Docstrings are considered part of the source code.
- Beginners often wrongly put docstrings after the function definition, or before.
- A docstring must be the first line of the function's body, or it won't count.
- The docstring must also be a string literal value. An octothorpe (#) won't count.

```
from bakery import assert_equal
def calculate_area(length: int, width: int) -> int:
    '''
    This function calculates the area of a rectangle based on its dimensions.
    Args:
        length (int): The length of the rectangle in feet.
        width (int): The width of the rectangle in feet.
    Returns:
        int: The area in square feet.
    '''
    area = length * width
    return area

assert_equal(calculate_area(5, 6), 30)
assert_equal(calculate_area(0, 6), 0)
assert_equal(calculate_area(4, 5), 20)
```

Check Your Understanding

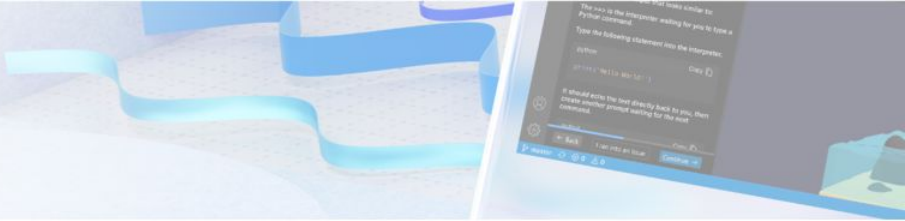


Question 1

Which of the following scenarios would be a good reason to document code?

- ❑ To test if a function has the right inputs for some given outputs.
- ❑ To make code easier to understand for other people.
- ❑ To keep a copy of code that you want to eliminate in case you need the code later.

Check Your Understanding

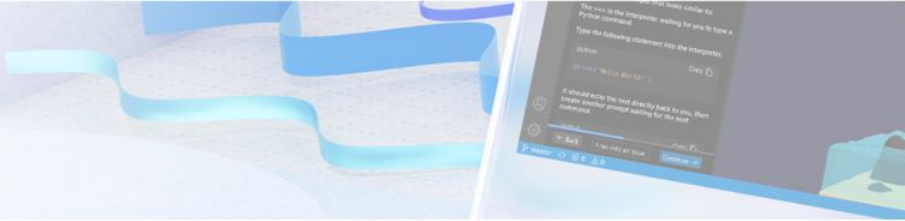


Question 1

Which of the following scenarios would be a good reason to document code?

- ❑ To test if a function has the right inputs for some given outputs.
- ✓ To make code easier to understand for other people.
  Documentation makes it easier for other people to read and understand your code.
- ❑ To keep a copy of code that you want to eliminate in case you need the code later.

Check Your Understanding

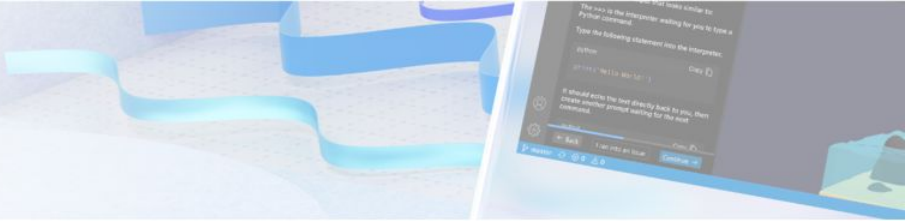


Question 3

Which statement is most accurate?

- ❑ Python completely ignores single-line comments when the code is executed.
- ❑ Python completely ignores multi-line and single-line comments when the code is executed.
- ❑ Python completely ignores multi-line comments when the code is executed.

Check Your Understanding



Question 3

Which statement is most accurate?

- ✓ Python completely ignores single-line comments when the code is executed.
  Unlike multi-line comments, Python completely ignores single-line comments during execution.

- ❑ Python completely ignores multi-line and single-line comments when the code is executed.

- ❑ Python completely ignores multi-line comments when the code is executed.

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

```
"""
add(a, b)

The sum of two numbers.

"""
```

There are many different standards of docstrings across languages, organizations, and teams. For assignments, we will specify which standard you should follow. Mostly this will be the numpy standard.

More Control Flow

More Control Flow - “Truthiness”

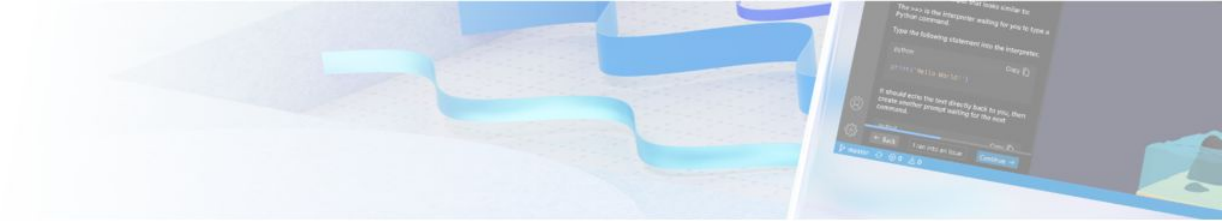
You may have noticed that occasionally, we don’t use explicit boolean expressions to evaluate if a statement is true or false. For example

if num: …

if num % 2: …

We call this the concept of *truthiness*

What Is Truthiness?

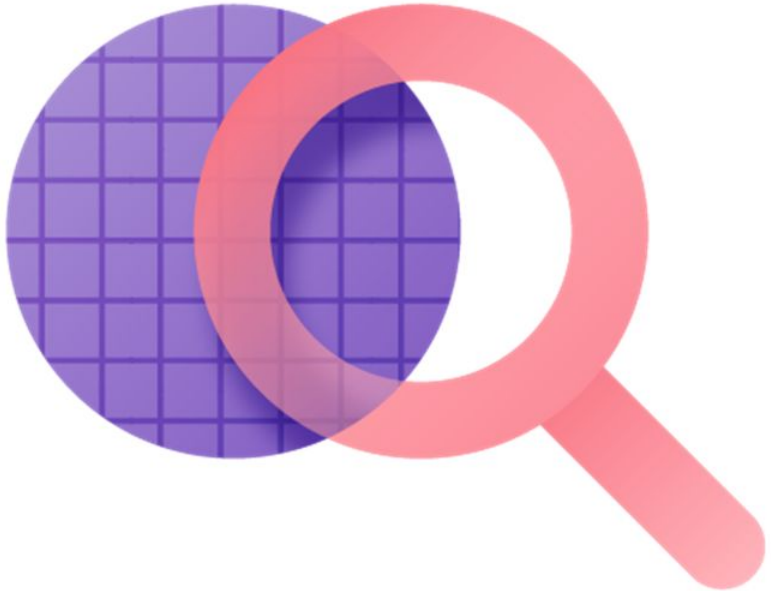


This table summarizes the truthiness rules for the primitive types:

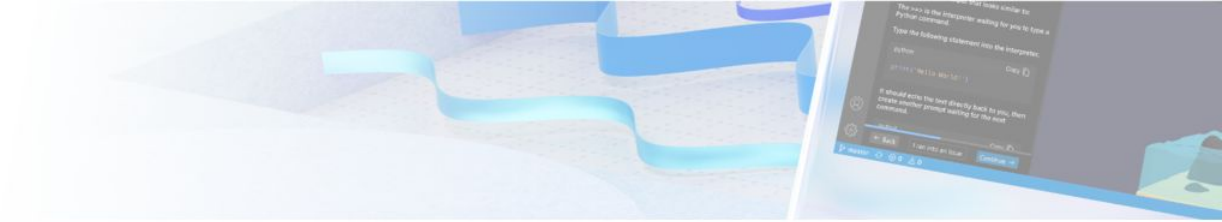
Type	Falsy	Truthy
Boolean	False	True
Integer	0	Non-zero number (for example, 5, -3)
Float	0.0	Non-zero number (for example, -3.0, 2.4)
String	""	Non-empty string (for example, "Hello")
None	None	

What Is Truthiness?

- Any expression can be evaluated as a conditional. *How* it's evaluated depends on its type.
- For integers and floats, any non-zero value is considered true.
- For strings, any non-empty string is considered true.
- The None type has no true values.



Testing Truthiness

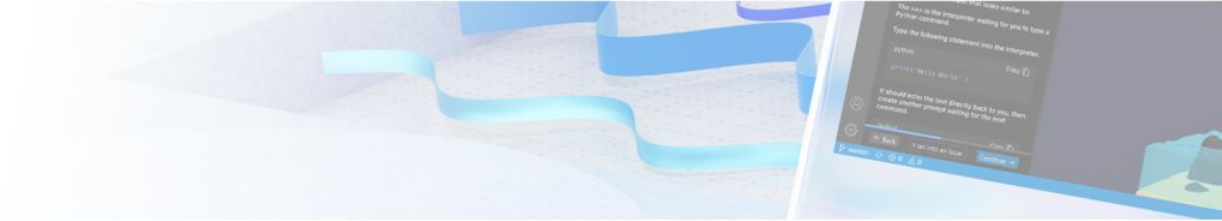


- These values aren't strictly equal (`==`) to `True` or `False`.

- A value's truthiness is based on whether it's considered `True` or `False` *when evaluated in the context of an* `if` *statement*.

```
if 5:
    print("This will be printed because 5 is Truthy")
else:
    print("This will not be printed!")
```

Testing Truthiness

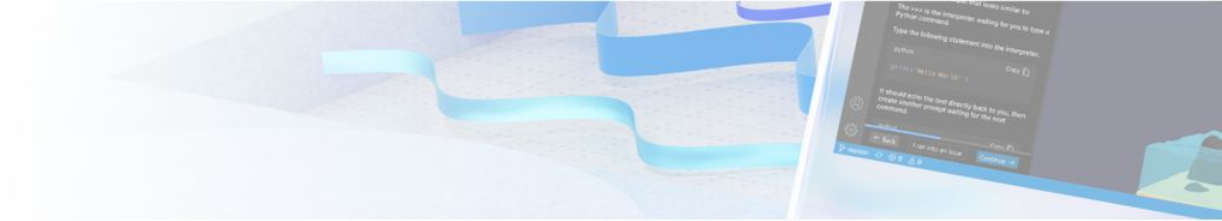


- Test for truthiness by converting the expression to a Boolean value by using the `bool` conversion function.

- The result will be `True` or `False`, indicating what the original expression's truthiness was.

```
print("The integer 5 is", bool(5))
print("The integer 0 is", bool(0))
print("The string 'Cat' is", bool('Cat'))
print("The string '' (the empty string) is", bool(''))
```

Testing Truthiness



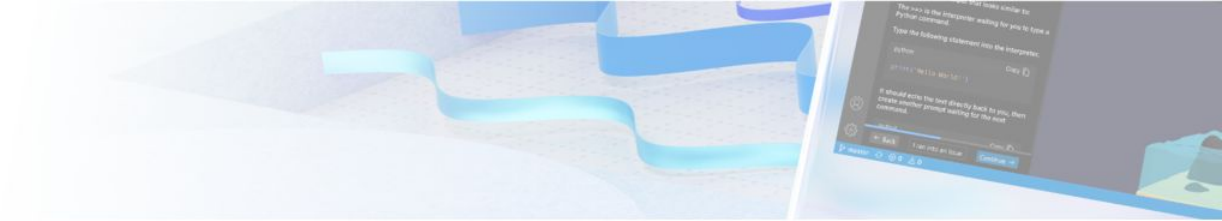
- The expressions themselves are not `True` or `False`, so you can say that the values are *truthy* and *falsy*.

- This distinguishes a *truthy* value (like 5) from the actual `True` Boolean value.

```
# This is False because True is not equal to 5
print(True == 5)

# This is True because 5 is Truthy
print(True == bool(5))
```

String Example

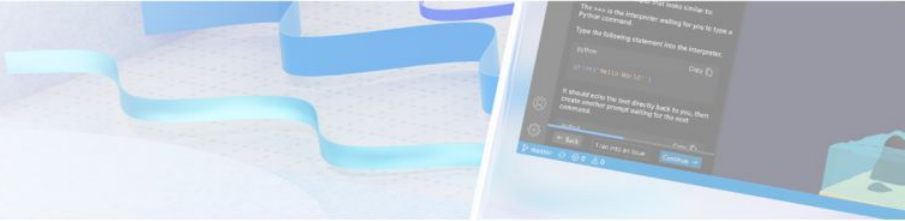


- **Example**: Taking input from a user.
- If the user enters an empty string, you'll print a different message.

```
name = input("What is your name?")

# `name` is a string variable
if name:
    # This path if `name` is NOT the empty string
    print("Your name is", name)
else:
    # This path if `name` is empty string
    print("No name given.")
```

Check Your Understanding

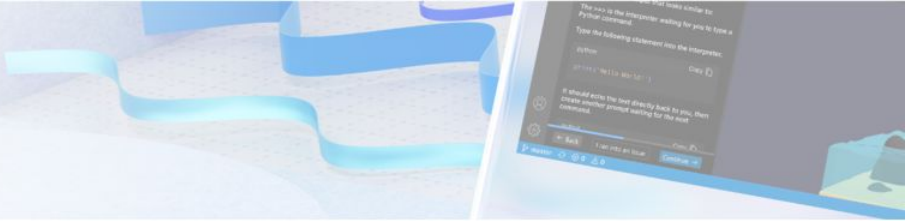


Question 1

Evaluate the following expression in terms of Truthiness: `"False"`

- ❑ Can’t be evaluated
- ❑ Falsy
- ❑ Truthy

Check Your Understanding

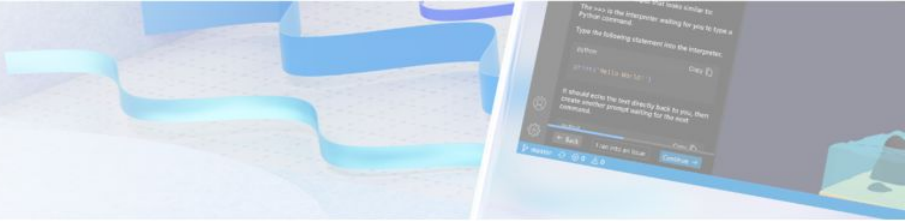


Question 1

Evaluate the following expression in terms of Truthiness: `"False"`

- ❑ Can't be evaluated
- ❑ Falsy
- ✓ Truthy
  This is a string containing the text `"False"`, which means it isn't empty and therefore *is* truthy.

Check Your Understanding

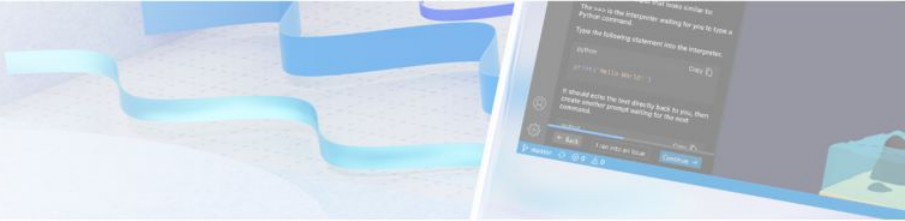


Question 2

Evaluate the following expression in terms of Truthiness: `8-4*2`

- ❑ Falsy
- ❑ Truthy
- ❑ Can't be evaluated

Check Your Understanding

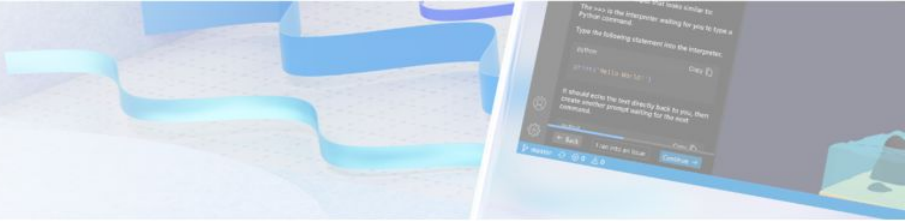


Question 2

Evaluate the following expression in terms of Truthiness: `8-4*2`

- ✓ Falsy
  This expression evaluates to 0, which is the Falsy value for integers.
- ❑ Truthy
- ❑ Can’t be evaluated

Check Your Understanding

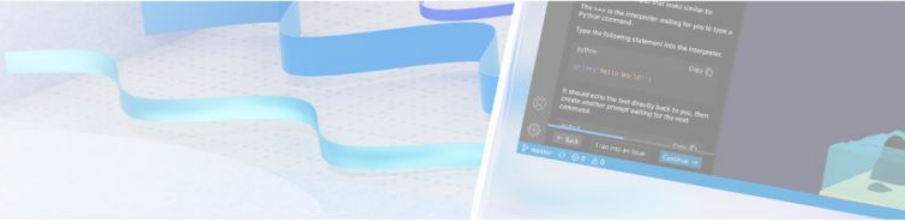


Question 3

Evaluate the following expression: `True != bool("Dog")`

- ❑ Can't be evaluated
- ❑ False
- ❑ True

Check Your Understanding



Question 3

Evaluate the following expression: `True != bool("Dog")`

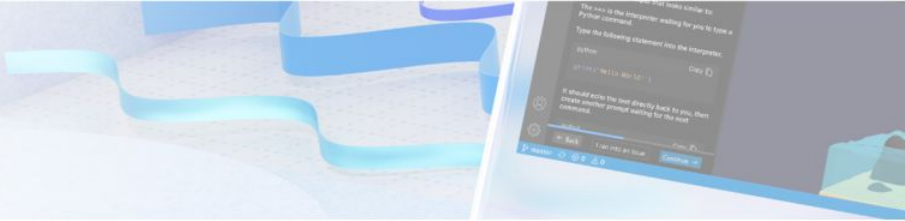
- ❑ Can't be evaluated

- ✓ False

  This is an inequality comparison between the Boolean literal value True and the result of converting a non-empty string to a Boolean. Because both values are True, the inequality results in False.

- ❑ True

Unnecessary Comparisons



```
if a_string != "":
    ...
# Simplifies to
if a_string:
    ...
#-----------------------
if a_number != 0:
    ...
# Simplifies to
if a_number:
    ...
```

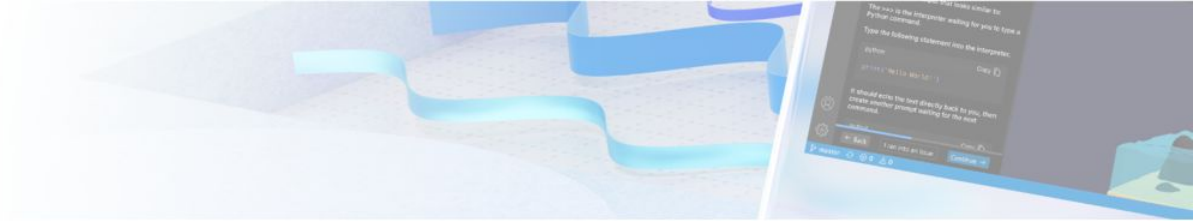
- Use truthiness when you need to check whether a string is empty, or a number isn't 0.
- **Example**: Each of these strings can become clearer and more concise.

Unnecessary Comparisons

Truthiness is useful when you want to check whether a string is empty or whether a number is zero before you divide.

```
dollar_amount = input("Enter a dollar amount")
if dollar_amount:
    # Need the string to be non-empty before we can index
    dollar_symbol = dollar_amount[0]
#------------------------
people = input("How many people are there?")
if people:
    # Need the string to be non-empty before we can convert
    number_of_people = int(people)
    if number_of_people:
        # need the integer to be non-zero before we can divide
        print("The price per person is:", 5/number_of_people)
```

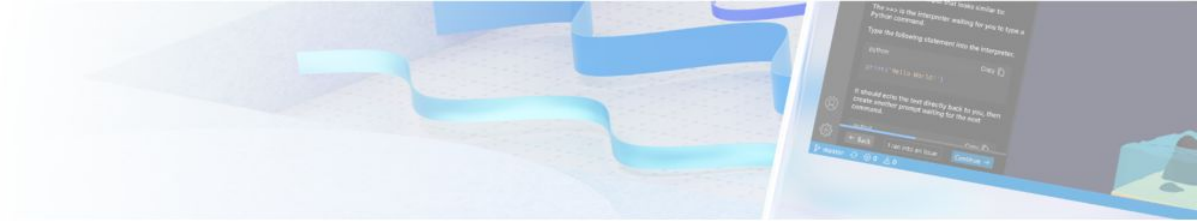
Accidental Truthiness



- **Example**: The variable `alpha` holds the value 5. However, the `or` operator has a lower priority than the `==` operator.
- Python first evaluates the equality test between the variable `alpha` and the value 3, which results in `False`.
- Because the operator is `or`, you evaluate its right-hand operand. Python evaluates the value 4 on its own and uses that value as the result of the conditional expression.

```
alpha = 5
# incorrect!
if alpha == 3 or 4:
    print("This WILL be printed!")
# correct!
if alpha == 3 or alpha == 4:
    print("This will NOT be printed!")
```

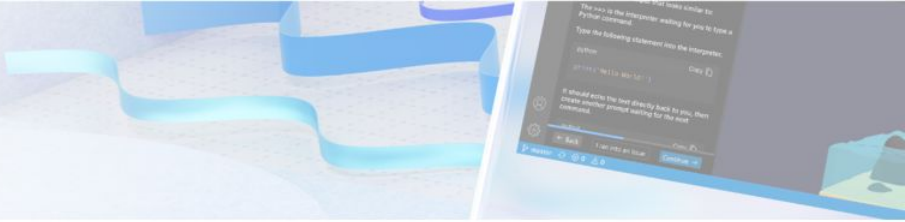
Accidental Truthiness



```
alpha = 5
# incorrect!
if alpha == 3 or 4:
    print("This WILL be printed!")
# correct!
if alpha == 3 or alpha == 4:
    print("This will NOT be printed!")
```

- Because 4 is a non-zero value, the entire expression evaluates to true.
- In the correct version of the program, you just repeat the comparison operator a second time.
- When evaluating expressions, keep track of whether truthiness was accidentally involved.

Check Your Understanding

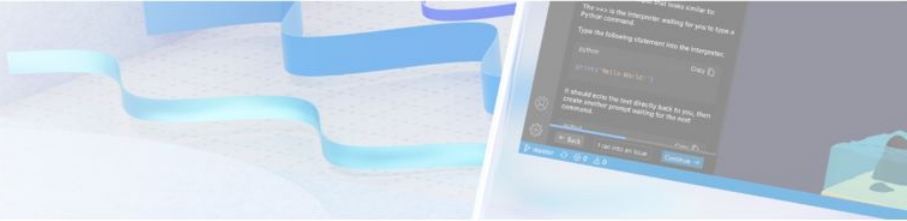


Question 4

Evaluate the following expression in terms of Truthiness: `-1 < 4 and 0`

- ❑ Falsy
- ❑ Truthy
- ❑ Can’t be evaluated

Check Your Understanding



Question 4

Evaluate the following expression in terms of Truthiness: `-1 < 4 and 0`

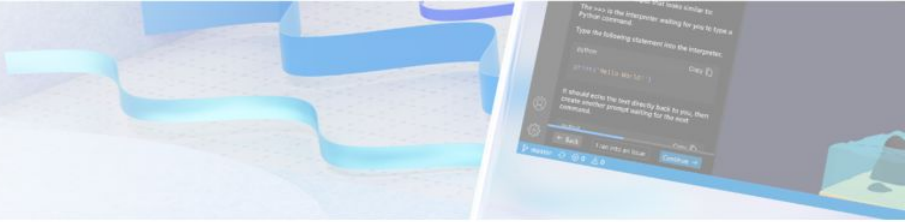
- ✓ Falsy

  This first checks whether `-1` is less than `4`. Afterwards, it checks the Truthiness of `0`, which is Falsy. Then it uses the `and` operator to combine the True and Falsy values, which results in a Falsy value.

- ❑ Truthy

- ❑ Can't be evaluated

Check Your Understanding

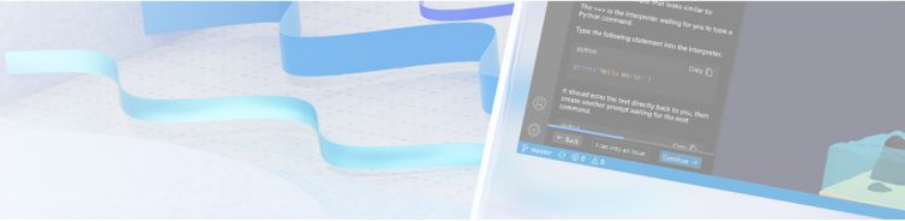


Question 5

Evaluate the following expression in terms of Truthiness: `1 > 5 or 3`

- ❑ Can't be evaluated
- ❑ Falsy
- ❑ Truthy

Check Your Understanding



Question 5

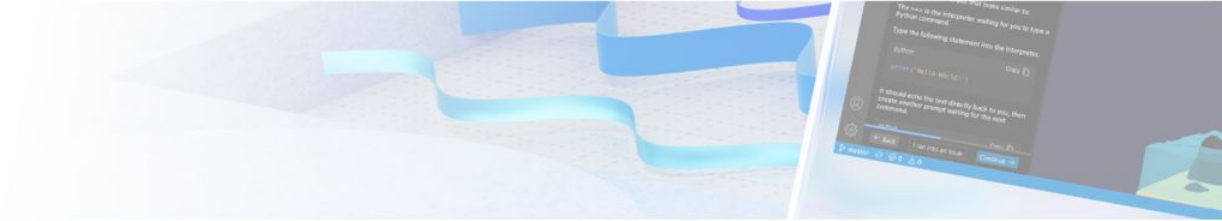
Evaluate the following expression in terms of Truthiness: `1 > 5 or 3`

- ❑ Can't be evaluated
- ❑ Falsy
- ✓ Truthy

  This first checks whether `1` is greater than `5`, which is False. Then it checks the Truthiness of `3`, which is Truthy. Then it uses the `or` operator to combine the False and Truthy values, which results in a Truthy value.

Nested If’s

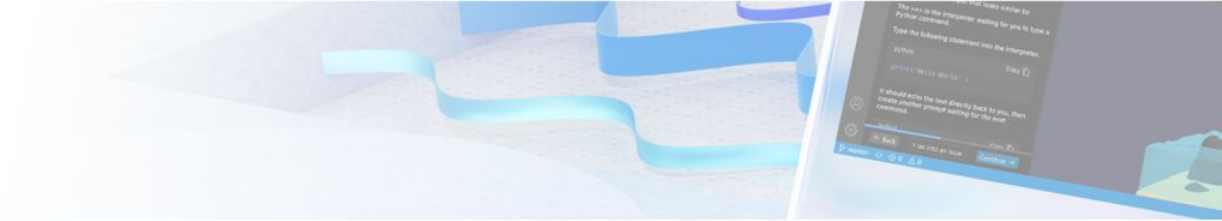
Introduction



- **Example**: In this code, the outer `if` statement has two branches.
- The inner `if` statement is on the True path of the outer `if` and has two branches of its own.
- The `else` body has no further branches.

```
age = 22
cost = 45
if age > 21:
    if cost < 10:
        print("Buy")
    else:
        print("Too expensive")
else:
    print("Too young")
```

Introduction

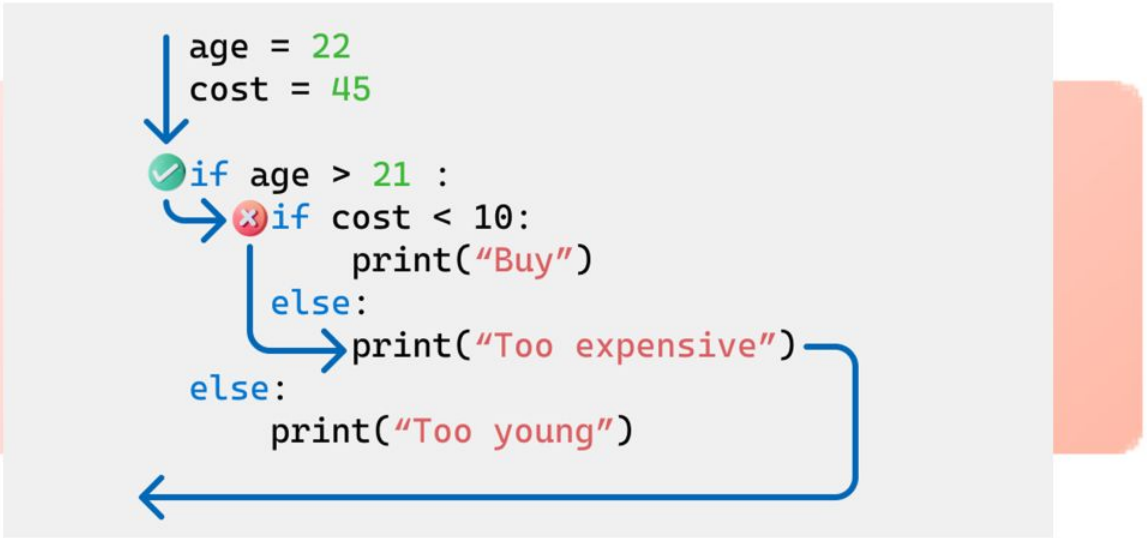


```
age = 22
cost = 45
if age > 21:
    if cost < 10:
        print("Buy")
    else:
        print("Too expensive")
else:
    print("Too young")
```

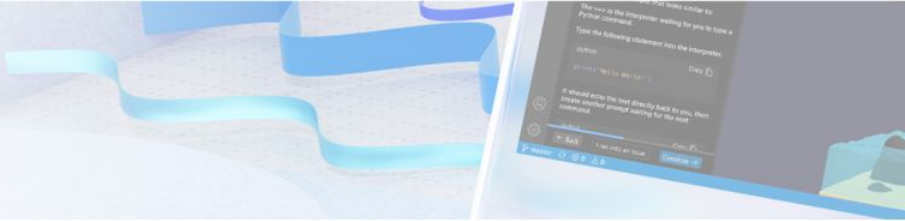
- **Example**: There are three branches in total because the first branch is divided into two parts.
- In other words, each new `if` adds only one new branch to the existing number of branches.

Introduction

This diagram overlays the paths of the branches.



Check Your Understanding



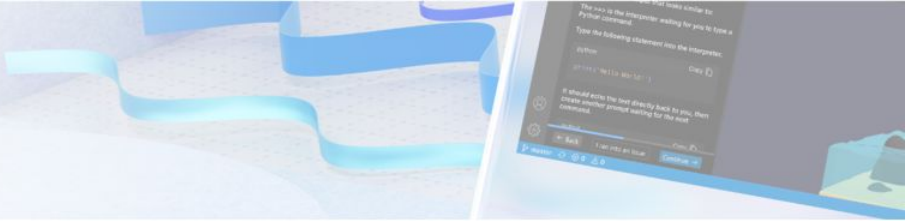
Question 1

How many branches does this code have?

```
if conditional:
  if second_conditional:
    print(5)
  print(1)
print(2)
```

- ❑ 0
- ❑ 3
- ❑ 2
- ❑ 1

Check Your Understanding



Question 1

How many branches does this code have?

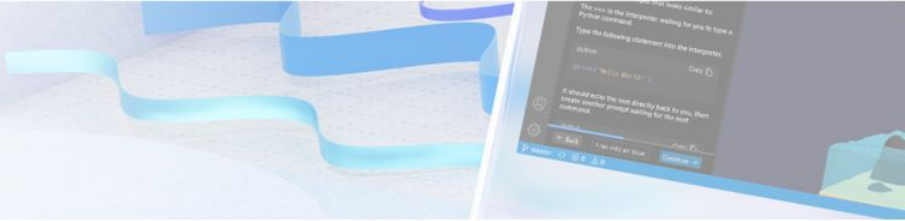
```
if conditional:
  if second_conditional:
    print(5)
  print(1)
print(2)
```

- ❑ 0
- ✓ 3

The outer `if` statement has two branches, and then the first branch turns into two further branches with the inner `if` statement.

- ❑ 2
- ❑ 1

Check Your Understanding



Question 2

Trace the code. What is the final value of the `salary` variable?

- ❑ 1000
- ❑ 0.0
- ❑ 500.0
- ❑ 250.0

```
salary = 1000
if salary > 100:
  taxes = .5
else:
  taxes = .1
salary = taxes * salary
if salary > 100:
  if salary < 500:
      taxes = 0
  salary = taxes * salary
```

Check Your Understanding

Question 2

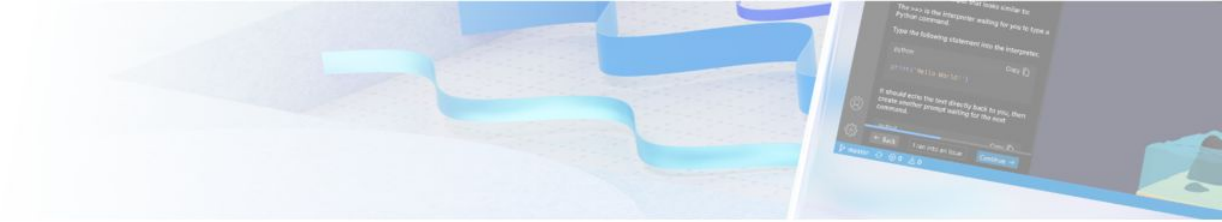
Trace the code. What is the final value of the `salary` variable?

```
salary = 1000
if salary > 100:
  taxes = .5
else:
  taxes = .1
salary = taxes * salary
if salary > 100:
  if salary < 500:
      taxes = 0
  salary = taxes * salary
```

- ❑ 1000
- ❑ 0.0
- ❑ 500.0
- ✓ 250.0

The first `if` statement causes the taxes to become .5, which in turn makes the salary 500.0. That causes the second `if` statement to trigger, but not the third. So, the salary is updated one more time to be 250.0.

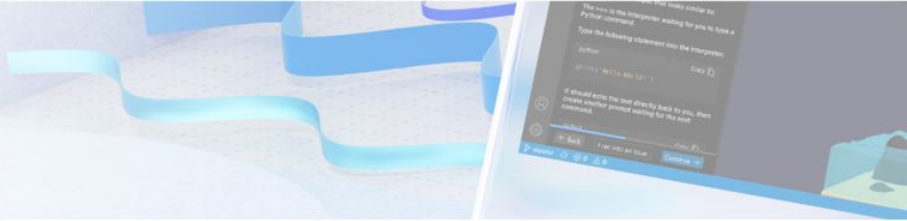
Number of Spaces



- When you nest a block of statements inside another block, the body of the block gets indented another 4 spaces.
- In this diagram, the numbers are on the left, and the Python code is on the right. Each level of nesting increases the number of space characters by 4.
- These are called the *whitespace rules*. The amount of whitespace controls what code is in what body.

```
0 → if age > 21 :
4 → ␣␣␣␣if cost < 10:
8 → ␣␣␣␣␣␣␣␣print("Buy")
4 → ␣␣␣␣else:
8 → ␣␣␣␣␣␣␣␣print("Too expensive")
0 → else:
4 → ␣␣␣␣print("Too young")
```

Check Your Understanding

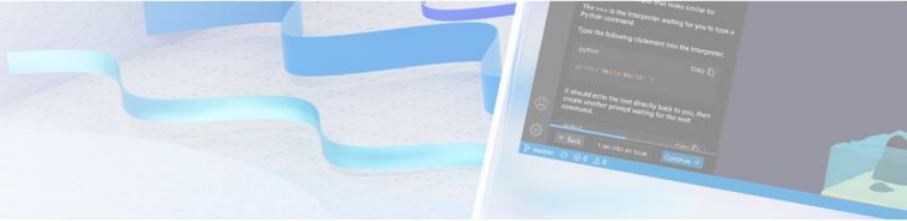


Question 3

If an `if` statement is nested inside another `if` statement, how many spaces will the body of the inner `if` statement have?

- ❑ 4
- ❑ 2
- ❑ 8

Check Your Understanding



Question 3

If an `if` statement is nested inside another `if` statement, how many spaces will the body of the inner `if` statement have?

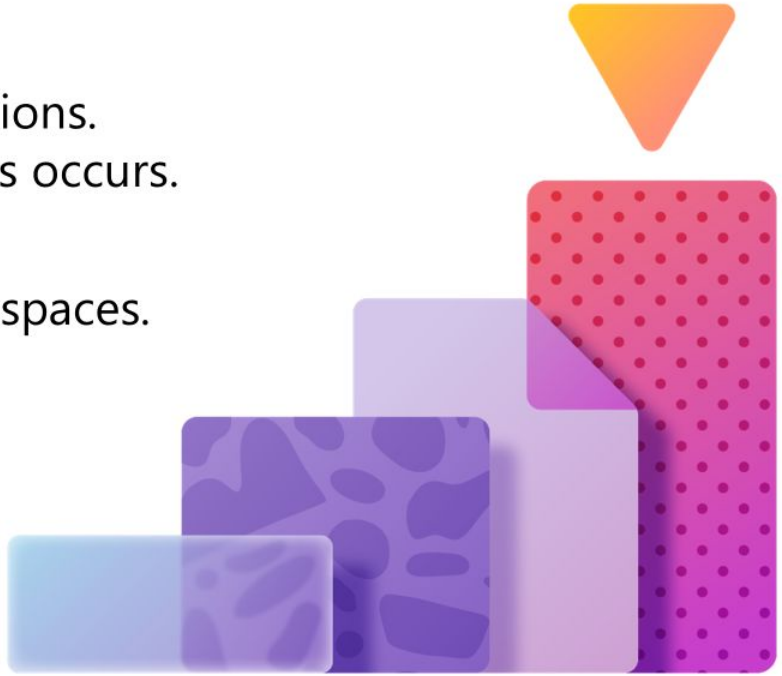
- ❑ 4
- ❑ 2
- ✓ 8

The outer `if` statement has 4 spaces in its body, and the inner `if` statement has another 4 spaces, for 8 spaces total.

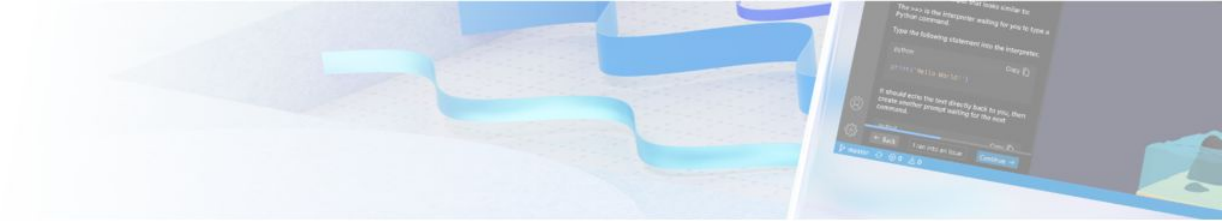
If and Functions

- You can put `if` statements inside of functions. Remember the whitespace rules when this occurs.

- Each nested block is indented another 4 spaces.



If and Functions



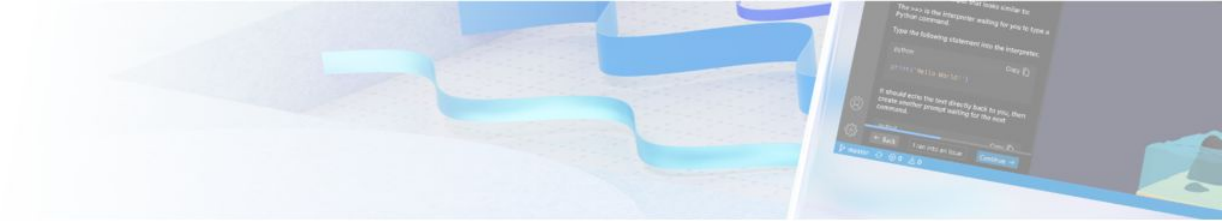
```
from bakery import assert_equal
def adjust_price(price: float, age: int) ->
float:
    if age > 60:
        return price * .8
    else:
        return price
assert_equal(adjust_price(5.75, 63), 4.60)
assert_equal(adjust_price(5.75, 45), 5.75)
```

- **Example**: In this function, the first line of the body is indented 4 spaces, but the second line of the body is indented 8 spaces.
- The first line isn't indented, so it comes after the body of the function.

What Goes Inside?

- It can be difficult to know whether a line of code belongs inside or outside a block.
- You must think critically about what you are trying to do with the `if` block.
- Remember, statements inside the `if` block are executed only if their conditional evaluates to True.

What Goes Inside?



```
cost = int(input("What is the cost?"))
# One line in body
if cost > 5:
    discount = 1
price = 2
# Two lines in body
if cost > 5:
    discount = 2
    price = 3
```

- **Example**: The first assignment of the price variable is *after* the `if` statement's body. The second assignment is *inside* the `if` statement.
- So, the first assignment will always be executed, but the second assignment might or might not be executed, *depending* on the value of cost.

ELIF Block

- The `elif` block is the same as an `else` block with an `if` statement inside.
- This type of block can be more convenient to write.

else + if

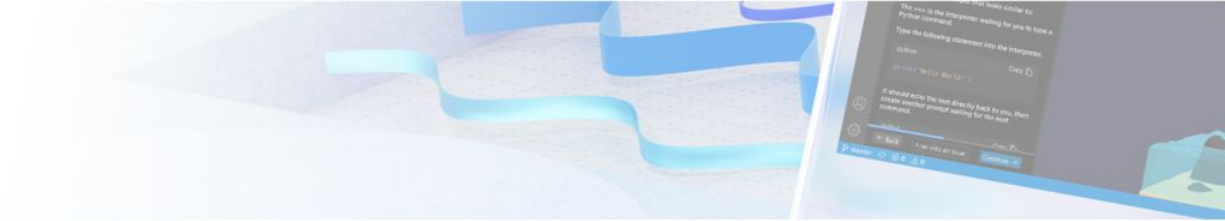
```
if "dog" in name:
    print("Is a dog")
else:
    if "cat" in name:
        print("Is a cat")
    else:
        print("Unknown")
```

elif

```
if "dog" in name:
    print("Is a dog")
elif "cat" in name:
    print("Is a cat")
else:
    print("Unknown")
```

Equivalent!

Two Ifs vs. ELSE If



if + if

```
if "dog" in name:
    print("Is a dog")
if "cat" in name:
    print("Is a cat")
```

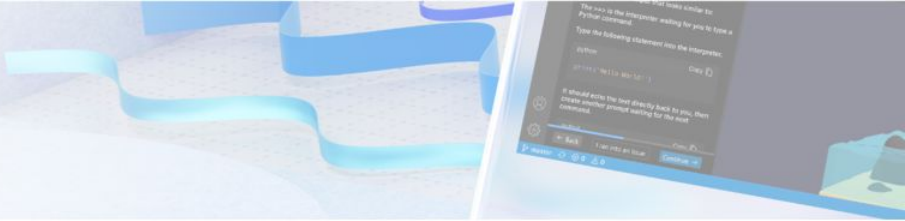
elif

```
if "dog" in name:
    print("Is a dog")
elif "cat" in name:
    print("Is a cat")
```

NOT Equivalent!

- The first piece of code has two `if` statements. Both `if` statements are always evaluated and potentially executed.
- The `elif` statement is evaluated only if the `if` statement evaluates to false.

Check Your Understanding

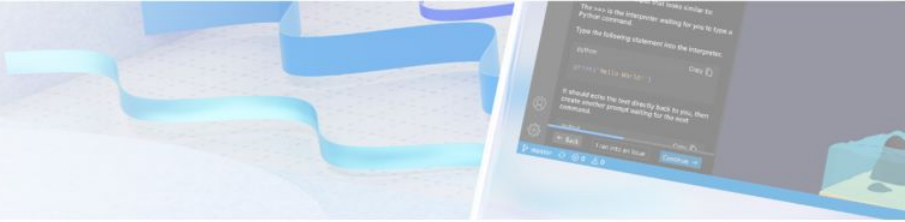


Question 4

Which of the following statements is most accurate?

- ❑ Every `if` statement must be followed by either an `else` or an `elif` statement.
- ❑ You can nest `if` statements inside other `if` statements, but not functions.
- ❑ The `elif` is equivalent to an `else` with an `if` statement nested inside.

Check Your Understanding



Question 4

Which of the following statements is most accurate?

- ❑ Every `if` statement must be followed by either an `else` or an `elif` statement.
- ❑ You can nest `if` statements inside other `if` statements, but not functions.
- ✓ The `elif` is equivalent to an `else` with an `if` statement nested inside.
  The `elif` is actually an `else` + `if` statement. This is how Python interprets it.

Common Mistakes

- Two kinds of mistakes are common with `if` statements:
  - Using an `if` statement when a conditional expression is fine on its own.
  - Testing if a Boolean expression is equal to True.

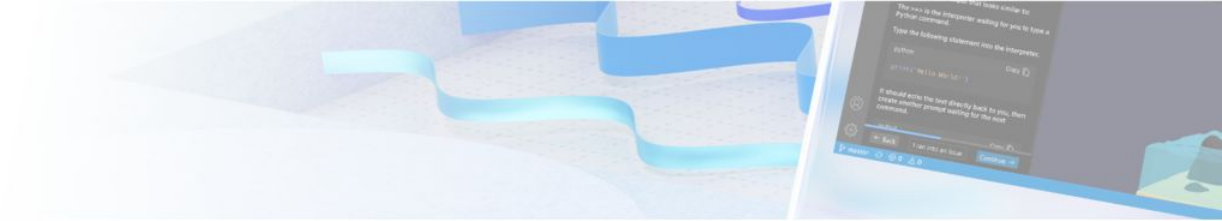
Unnecessary If

- **Example**: This function definition returns True if the parameter is greater than 5 or returns False.
- The conditional expression already evaluates to either True or False, so it was unnecessary to use an `if` statement.
- You can directly return the result of the conditional expression.
- You don't need to use `if` statements if you're just returning True and False.

```
def check_senior(age: int) -> bool:
    # This `if` statement is unnecessary!
    if age > 60:
        return True
    else:
        return False

# Exactly equivalent to
def check_senior(age: int) -> bool:
    return age > 60
```

Unnecessary Test



- **Example:** the expression `age >= 5 == True` is redundant in Python.
- `age > 5` already evaluates to either `True` or `False`.
- If you compare a Boolean value to `True`, then the result is the same Boolean value.
- `True` would become `True`, and `False` would become `False`.

```
def check_toddler(age: int):
    # The `== True` part is redundant!
    return (age >= 5) == True

def check_toddler(age: int):
    # Much better!
    return age >= 5
```

In-Class Lab

Lab - GitHub Module

For the remaining lab time, break into your pod groups and complete the following modules:

- **VSCode Unit 3: Lesson 1 - Lesson 4**
- **VSCode Unit 2: Lesson 6 - Lesson 8**

Wrap-Up

Lab (Due 03/28)

Taipei City, Taiwan

The company you work for, Seng-Links, aims to identify periods when a user sleeps or exercises using their varying recorded heart rates.

Your company has provided you a data folder (*data/*) of **5 files** that contain heart-rate samples from a participant. The participants device records heart rate data every 5 minutes (aka *sampling rate*).

You are tasked with writing code that **processes each data file**. You will utilize test-driven development in order to complete this project.

Tuesday...

On Tuesday we will expand our exploration of control flow in Python to include:

- Logical conditional patterns
- For loops
- For loop common patterns

Jupyter: scratchpad of the data scientist

If you understand what you're doing, you're not learning anything. - Anonymous

Glossary