

Labs

 Collaboration is encouraged

In a pair,

Working together on the whole lab, not
dividing it up, and

Including both partners' names on the
submission.

Assignments

 No collaboration is allowed

Assignments need to be your own work,

But you can get help from the coaches
and from me.

It's a violation of academic integrity to submit anyone else's work as your own.

That's true whether it's from a friend in the class, something you find online, or written by AI.

It's plagiarism even if you change things or
“paraphrase” the code.

It's much better to turn an assignment in late,
incomplete, or even not to turn it in at all than it is
to use someone else's work!

→ Notebook: *Where are we?*

Defining and evaluating functions

Recall how you used functions in middle-school math:

$$\text{Given } f(x) = |x| + 2$$

$$f(-3) = |-3| + 2$$

$$= 3 + 2$$

$$= 5$$



*The parameter x stands
for varying values*

Python functions work much the same way:

```
def f(x): return abs(x) + 2
```

Python functions work much the same way:

```
def f(x): return abs(x) + 2
```

```
f(-3)
```

Python functions work much the same way:

```
def f(x): return abs(x) + 2
```

```
f(-3)
```

Directory

<i>Directory</i>	
<i>Name</i>	<i>Value</i>
<i>x</i>	-3

Python functions work much the same way:

```
def f(x): return abs(x) + 2
```

f(-3)
→ abs(x) + 2

Directory	
Name	Value
x	-3

Python functions work much the same way:

```
def f(x): return abs(x) + 2
```

f(-3)

→ abs(x) + 2

→ abs(-3) + 2

Directory

<i>Directory</i>	
<i>Name</i>	<i>Value</i>
<i>x</i>	-3

Python functions work much the same way:

```
def f(x): return abs(x) + 2
```

f(-3)

→ abs(x) + 2

→ abs(-3) + 2

→ 3 + 2

Directory

<i>Directory</i>	
<i>Name</i>	<i>Value</i>
<i>x</i>	-3

Python functions work much the same way:

```
def f(x): return abs(x) + 2
```

f(-3)

→ abs(x) + 2

→ abs(-3) + 2

→ 3 + 2

→ 5

Directory

<i>Directory</i>	
<i>Name</i>	<i>Value</i>
<i>x</i>	-3

Example

Mary Berry needs to know how many cakes to bake for her cake shop.

To avoid running out or having too many, she wants to bake two cakes more than the number she sold the previous day.

E.g., if Mary sells eight cakes on Monday, she makes ten cakes on Tuesday.

Let's write some code to help Mary!



```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

*Keyword to **define** a function*



```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

Function name

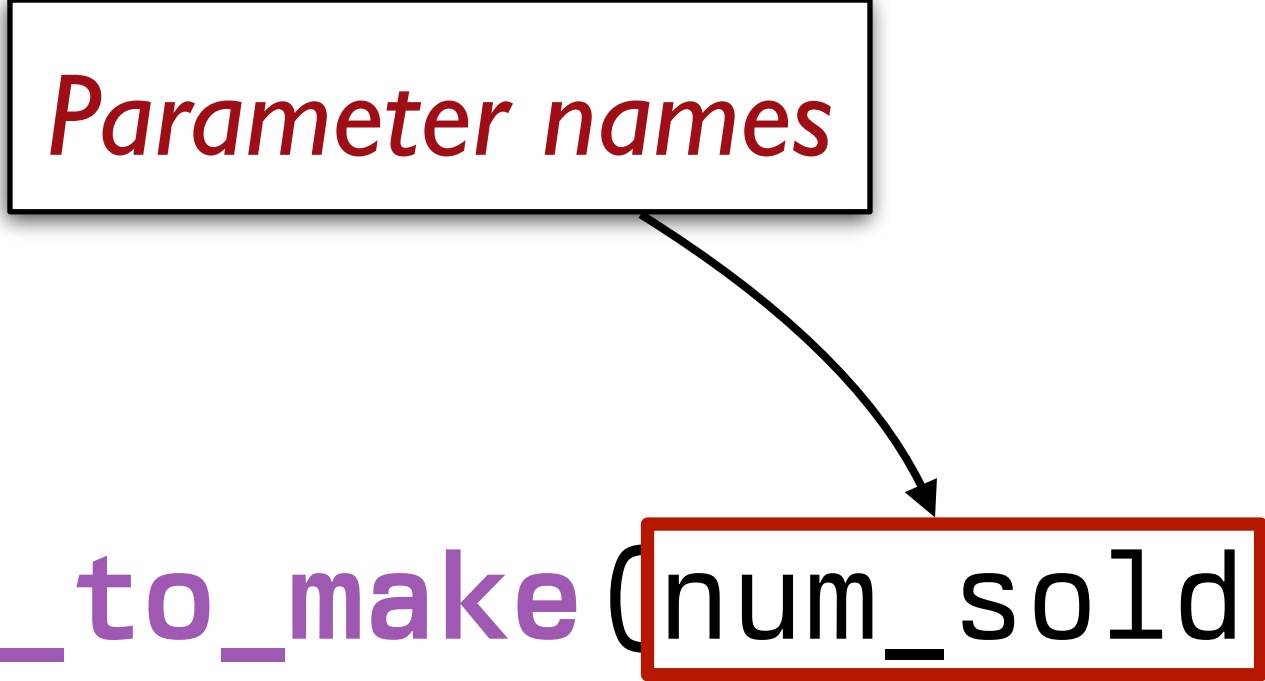


```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

Parameter names

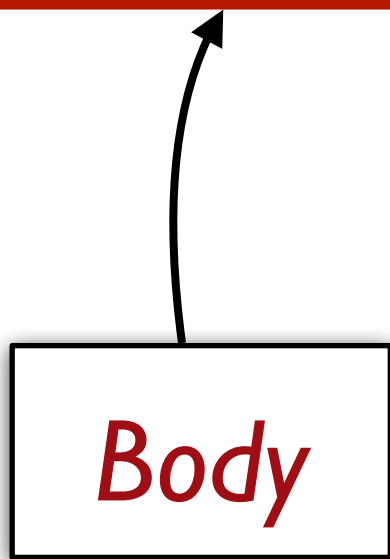
```
def cakes_to_make(num_sold):  
    return num_sold + 2
```



```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

Body



A diagram illustrating the components of a function definition. A box labeled "Body" in red italicized font is positioned below the function body code. A curved arrow points from the box to the return statement "return num_sold + 2", which is highlighted with a red rectangular border in the code above.

```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

Note that the parameter names are only defined inside the function body:

```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

```
cakes_to_make(10)
```



12

```
num_sold
```



Error!

Once the function is finished, the names are removed from the directory.

Calling a function

```
yesterday = 10
already_made = 5

to_make = (
    cakes_to_make(yesterday)
    - already_made
)
```

<i>Directory</i>	
<i>Name</i>	<i>Value</i>

Calling a function

```
yesterday = 10  
already_made = 5  
  
to_make = (  
    cakes_to_make(yesterday)  
    - already_made  
)
```

Directory	
Name	Value
<i>yesterday</i>	10

Calling a function

```
yesterday = 10  
already_made = 5  
  
to_make = (  
    cakes_to_make(yesterday)  
    - already_made  
)
```

Directory	
Name	Value
<i>yesterday</i>	10
<i>already_made</i>	5

Calling a function

```
yesterday = 10  
already_made = 5  
  
to_make = (  
    cakes_to_make(yesterday)  
    - already_made  
)
```

Directory	
Name	Value
<i>yesterday</i>	10
<i>already_made</i>	5
<i>to_make</i>	

Calling a function

```
yesterday = 10  
already_made = 5
```

```
to_make = (  
    cakes_to_make(yesterday)  
    - already_made  
)
```

Directory	
Name	Value
<i>yesterday</i>	10
<i>already_made</i>	5
<i>to_make</i>	

Calling a function

```
yesterday = 10  
already_made = 5  
  
to_make = (  
    cakes_to_make(10)  
    - already_made  
)
```

Directory	
Name	Value
<i>yesterday</i>	10
<i>already_made</i>	5
<i>to_make</i>	

Calling a function

ye
a
to
)

```
def cakes_to_make(num_sold):  
    return num_sold + 2
```

Directory	
Name	Value
num_sold	10

Calling a function

ye
a
to
)

```
def cakes_to_make(num_sold):  
    return 10 + 2
```

Directory	
Name	Value
num_sold	10

Calling a function

ye
a
to
)

```
def cakes_to_make(num_sold):  
    return 12
```

Directory	
Name	Value
num_sold	10

Calling a function

```
yesterday = 10  
already_made = 5  
  
to_make = (  
    12  
    - already_made  
)
```

Directory	
Name	Value
<i>yesterday</i>	10
<i>already_made</i>	5
<i>to_make</i>	

Calling a function

```
yesterday = 10  
already_made = 5  
  
to_make = (  
    12  
    - 5  
)
```

Directory	
Name	Value
<i>yesterday</i>	10
<i>already_made</i>	5
<i>to_make</i>	

Calling a function

```
yesterday = 10  
already_made = 5  
  
to_make = 7
```

Directory	
Name	Value
<i>yesterday</i>	10
<i>already_made</i>	5
<i>to_make</i>	

Calling a function

```
yesterday = 10  
already_made = 5
```

```
to_make = 7
```

Directory	
Name	Value
yesterday	10
already_made	5
to_make	7

We say a parameter name has only *local scope*, while names defined outside a function have *global scope*.

Formatting matters! A line of code is only part of the body of a function if it's indented:

```
def cakes_to_make(num_sold):  
    tomorrow = num_sold + 2  
return tomorrow
```

Error!

Formatting matters! A line of code is only part of the body of a function if it's indented:

```
def cakes_to_make(num_sold):  
    tomorrow = num_sold + 2  
    return tomorrow
```

*Now this line is part of function,
so `tomorrow` is defined!*

Functions are abstractions over specific computations

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus $\frac{1}{4}$ of the preparation time in minutes.

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus $\frac{1}{4}$ of the preparation time in minutes.

Chocolate cake

Ingredients: \$10

Prep. time: 20 min.

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus $\frac{1}{4}$ of the preparation time in minutes.

Chocolate cake

Ingredients: \$10

Prep. time: 20 min.

$$\text{choc_cake_price} = (2 * 10) + (1/4 * 20)$$

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus $\frac{1}{4}$ of the preparation time in minutes.

Chocolate cake

Ingredients: \$10

Prep. time: 20 min.

$$\text{choc_cake_price} = (2 * 10) + (1/4 * 20)$$

Cheesecake

Ingredients: \$15

Prep. time: 36 min.

For Mary's cake shop, we want to determine the price of each cake based on the cost of the ingredients and the time to prepare it.

As the price, she uses twice the cost of the ingredients plus $\frac{1}{4}$ of the preparation time in minutes.

Chocolate cake

Ingredients: \$10

Prep. time: 20 min.

$$\text{choc_cake_price} = (2 * 10) + (1/4 * 20)$$

Cheesecake

Ingredients: \$15

Prep. time: 36 min.

$$\text{cheesecake_price} = (2 * 15) + (1/4 * 36)$$

We use functions to avoid repetitive code when we need to perform the same operations on different values.

choc_cake_price = (2 * 10) + (1/4 * 20)

cheesecake_price = (2 * 15) + (1/4 * 36)

We use functions to avoid repetitive code when we need to perform the same operations on different values.

$$\textit{choc_cake_price} = (2 * 10) + (1/4 * 20)$$

$$\textit{cheesecake_price} = (2 * 15) + (1/4 * 36)$$

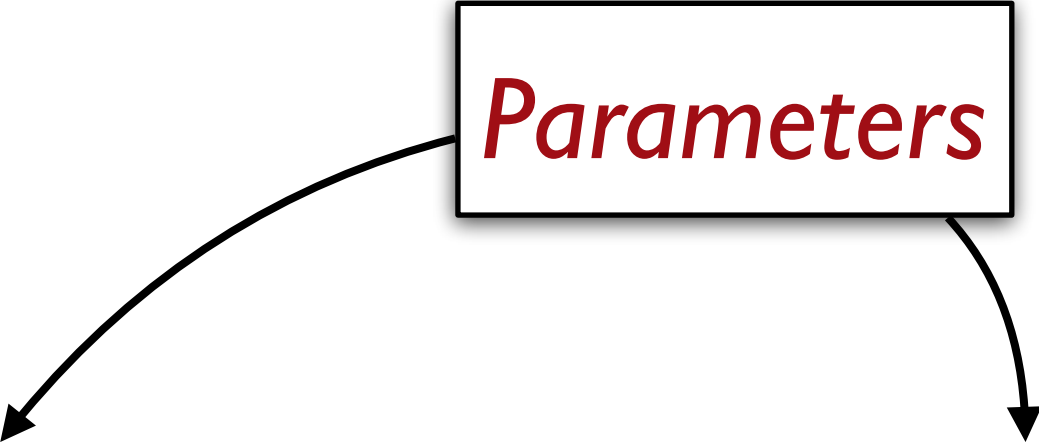
$$(2 * \text{ingredients_cost}) + (1/4 * \text{prep_time})$$

We use functions to avoid repetitive code when we need to perform the same operations on different values.

$$\text{choc_cake_price} = (2 * 10) + (1/4 * 20)$$

$$\text{cheesecake_price} = (2 * 15) + (1/4 * 36)$$

```
def cake_price(ingredients_cost, prep_time):  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

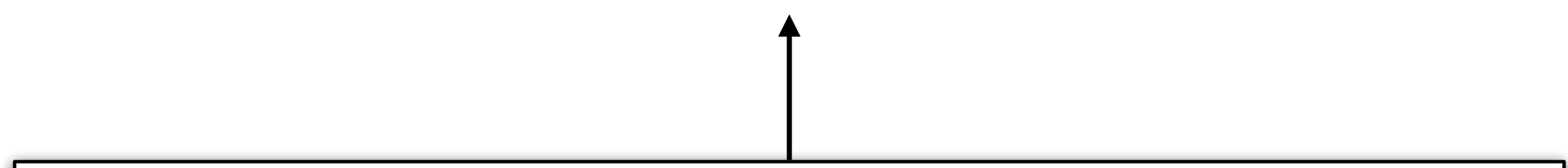


```
def cake_price(ingredients_cost, prep_time):  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

*The **parameters** are the values passed into the function that it needs to know for each operation.*

```
def cake_price(ingredients_cost, prep_time):  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

Statement computed each time the function is called

A rectangular box with a black border and a light gray shadow is positioned below the code. Inside the box, the text "Statement computed each time the function is called" is written in a red, italicized font. A black arrow points vertically upwards from the top center of the box to the line of code containing the expression (2 * ingredients_cost) + (1/4 * prep_time) in the return statement.

```
def cake_price(ingredients_cost, prep_time):  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

Price of chocolate cake

```
cake_price(10, 20)
```

Price of cheesecake

```
cake_price(15, 36)
```

To calculate the price of chocolate cake or cheesecake, you just call your function and pass in the relevant values!

Improving our function definitions

```
def c(x, y):  
    return (2 * x) + (1/4 * y)
```

*If you're looking at someone else's notebook and you see this function, you'd have no idea why they wrote it. What is this function **used for**? What are **x** and **y**?*

```
def cake_price(ingredients_cost, prep_time):  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

Just using good names goes a long way!

```
def cake_price(  
    ingredients_cost: float,  
    prep_time: float  
):  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

We specify the **type** of each parameter so that other people – or our future selves – know what kind of values the function expects.

✓ cake_price(10.70, 2.5)

✗ cake_price("expensive", "slow")

```
def cake_price(  
    ingredients_cost: float,  
    prep_time: float  
) -> float:  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

*And we can specify the type of value the function **returns**.*

Commands

+ Code + Text ▶ Run all

✓ RAM
Disk

⌵

👤 ⚙️ ⌵

☰

🔍

⏏

🔑

📁

[1]
✓ 0s

def **cake_price**(ingredients_cost: float, prep_time: float) -> float:
| return (2 * ingredients_cost) + (1/4 * prep_time)

[2]
✓ 0s

cake_price(10.70, 2.5)

➡ 22.025

[]

cake_price("expensive", "slow")

[]

▶ cake_price(10.70, 2.5) + "is the price"

⬆ ⬇ 🔗 💬 ⚙️ 📄 🗑️ ⋮

Colab is warning us
that we're using the
function wrong!

{ } Variables

📄 Terminal

✓ 10:37 PM

📄 Python 3

```
def cake_price(  
    ingredients_cost: float,  
    prep_time: float  
) -> float:  
    """Calculate price of cake based on ingredient  
    cost and preparation time.  
    """  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

Additionally, a *docstring* explains what the function does.

Practice

```
def triangle_area(b, h):  
    return 1/2 * b * h
```

How can we improve this?

```
def triangle_area(base: float, height: float) -> float:  
    """Return the area of the given triangle."""  
    return 1/2 * base * height
```

Testing functions

```
def cake_price(  
    ingredients_cost: float,  
    prep_time: float  
) -> float:  
    """Calculate price of cake based on ingredient  
    cost and preparation time.  
    """  
    return (2 * ingredients_cost) + (1/4 * prep_time)
```

*Our function looks good, but does it **work** correctly? We should test it!*

```
def cake_price(  
    ingredients_cost: float,  
    prep_time: float  
) -> float:  
    """Calculate price of cake based on ingredient  
    cost and preparation time.  
    """  
    return (2 * ingredients_cost) + (1/4 * prep_time)  
  
# Price of chocolate cake  
assert cake_price(10, 20) == 25  
# Price of cheesecake  
assert cake_price(15, 36) == 39
```

Our function looks good, but does it **work** correctly? We should test it!

colab.research.google.com/drive/1utr2-kI2Kkz5IIKSy0SV-x

Commands + Code + Text ▶ Run all

✓ RAM Disk

⌵ 👤 ⚙️ ⌵

⋮ 🔍 <> 🔗 📁 📊

[1]
✓ 0s ▶

```
def cake_price(ingredients_cost: float, prep_time: float) -> float:
    """Calculate price of cake based on ingredient cost and preparation time."""
    return (2 * ingredients_cost) + (1/4 * prep_time)

# Price of chocolate cake
assert cake_price(10, 20) == 25
# Price of cheesecake
assert cake_price(15, 36) == 39
```

⬆ ⬇ 🗑 ⋮

Variables Terminal

✓ 12:49 PM 📄 Python 3

All good!

colab.research.google.com/drive/1utr2-kl2Kkz5lIKSy0SV-x

Commands + Code + Text ▶ Run all

✓ RAM Disk

⌵ 👤 ⚙ ⌵

⋮

📄

<>

🔍

🔑

📁

📊

[2] 0s

▶

```
def cake_price(ingredients_cost: float, prep_time: float) -> float:
    """Calculate price of cake based on ingredient cost and preparation time."""
    return (2 * ingredients_cost) + (1/3 * prep_time)

# Price of chocolate cake
assert cake_price(10, 20) == 25
# Price of cheesecake
assert cake_price(15, 36) == 39
```

⌵

```
... -----
AssertionError                                Traceback (most recent call last)
/tmp/ipykernel_2667/2318233207.py in <cell line: 0>()
      4
      5 # Price of chocolate cake
----> 6 assert cake_price(10, 20) == 25
      7 # Price of cheesecake
      8 assert cake_price(15, 36) == 39

AssertionError:
```

Uh oh.

{ } Variables 📄 Terminal

⌚ 12:50 PM 📄 Python 3

Free shipping for orders over \$75

DARKSIDE RECORDS

EST. 2010

FOUCHKEEPSIE, NY

All products

David Bowie

Log in

Cart (\$0.00)

Hide filters

322 Results

In stock only (204)

Price

\$ 0 to \$ 399

The highest price is \$399.98

Format

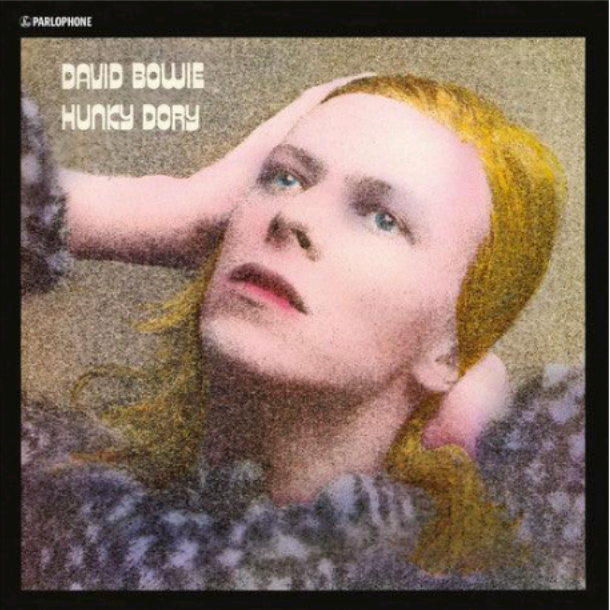
Books (5)

New 7/Rock (1)

New BluRay/Criterion (1)

Rewards (1)

New CD/Classical (1)



David Bowie- Hunky Dory

New Vinyl

\$24.99



David Bowie- Station To Station

New Vinyl

\$24.99



David Bowie- Rise and Fall Of Ziggy Stardust an...

New Vinyl

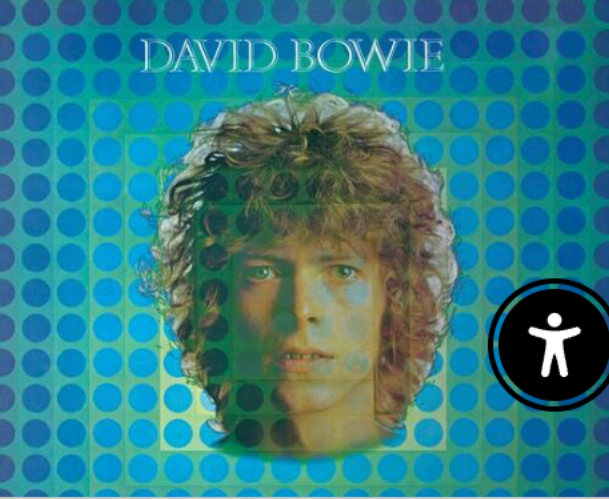
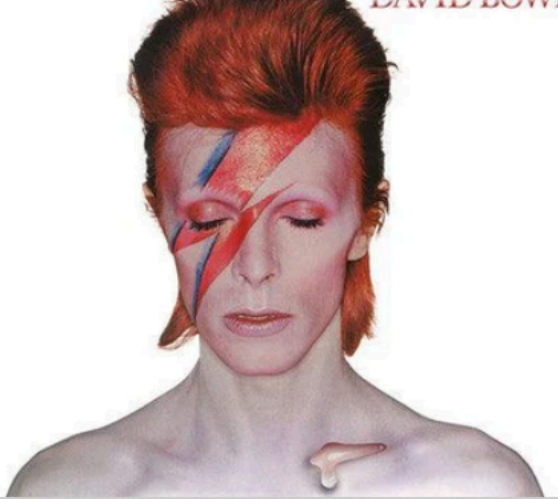
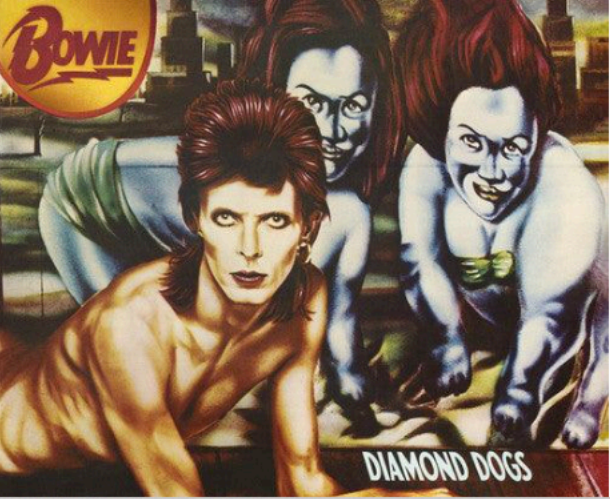
\$26.99



David Bowie- Labyrinth Soundtrack

New Vinyl

\$29.99



Free shipping for orders over \$75

DARKSIDE RECORDS

EST. 2010

POUGHKEEPSIE, NY

All products

David Bowie

Log in

Cart (\$0.00)

Hide filters

322 Results

Sort: Relevance

In stock only (204)

Price

\$ 0 to \$ 399

The highest price is \$399.98

Format

Books (5)

New 7/Rock (1)

New BluRay/Criterion (1)

Rewards (1)

New CD/Classical (1)

David Bowie- Hunky Dory

New Vinyl

\$24.99

David Bowie- Station To Station

New Vinyl

\$24.99

David Bowie- Rise and Fall Of Ziggy Stardust and The Spiders From Mars

New Vinyl

\$26.99

David Bowie- Labyrinth Soundtrack

New Vinyl

\$29.99

→ Notebook: *Testing functions*

Practice

```
def triangle_area(base: float, height: float) -> float:  
    """Return the area of the given triangle."""  
    return 1/2 * base * height
```

What tests should we add?

```
def triangle_area(base: float, height: float) -> float:  
    """Return the area of the given triangle."""  
    return 1/2 * base * height  
  
assert triangle_area(10, 10) == 50  
assert triangle_area(1, 5) == 2.5
```

Functions with comparisons and conditional statements

→ Notebook: *Functions with comparisons and conditional statements*

```
def is_wide(width: int, height: int) -> bool:
    """Return True if the image is wider than it is
    tall.
    """
    return width > height

assert is_wide(20, 10) == True
assert is_wide(10, 20) == False
```

```
def image_format(width: int, height: int) -> str:
    """Return whether the image is landscape (wide)
    or portrait (tall).
    """
    if width > height:
        return "landscape"
    else:
        return "portrait"

assert image_format(20, 10) == "landscape"
assert image_format(10, 20) == "portrait"
```

```
def image_format(width: int, height: int) -> str:
    """Return whether the image is landscape (wide)
    or portrait (tall).
    """
    if width > height:
        return "landscape"
    else:
        return "portrait"

assert image_format(20, 10) == "landscape"
assert image_format(10, 20) == "portrait"
```

What's wrong with this code?

```
def image_format(width: int, height: int) -> str:
    """Return whether the image is landscape (wide),
    square, or portrait (tall).
    """
    if width > height:
        return "landscape"
    elif width == height:
        return "square"
    else:
        return "portrait"

assert image_format(20, 10) == "landscape"
assert image_format(10, 20) == "portrait"
```

```
def image_format(width: int, height: int) -> str:
    """Return whether the image is landscape (wide),
    square, or portrait (tall).
    """
    if width > height:
        return "landscape"
    elif width == height:
        return "square"
    else:
        return "portrait"

assert image_format(20, 10) == "landscape"
assert image_format(10, 10) == "square"
assert image_format(10, 20) == "portrait"
```

Practice

Exercise

Write a handful of **assert** statements to test the following function:

```
def letter_grade(score: int | float) -> str:
    """Given a score between 0 and 100, returns
    the letter grade of:
    - "A" if the score is 90 or greater,
    - "B" if the score is in the 80s,
    - "C" if the score is lower than 80.
    """
    ...
```

Which versions pass all of the tests?

```
def letter_grade(score):  
    if score >= 80:  
        return "B"  
    elif score >= 90:  
        return "A"  
    else:  
        return "C"
```

```
def letter_grade(score):  
    if score > 90:  
        return "A"  
    elif score > 80:  
        return "B"  
    else:  
        return "C"
```

```
def letter_grade(score):  
    if score >= 90:  
        return "A"  
    elif score >= 80:  
        return "B"  
    else:  
        return "C"
```

Exercise

A year is a leap year if:

The year is divisible by 4 but not divisible by 100, or

The year is divisible by 400.

Complete the following function:

```
def is_leap_year(year):  
    """Return True if year is a leap year."""  
    ...
```

*You can use the % operator to check if year
is divisible by 4: `year % 4 == 0`*

