Authors' note: The following is the instructional material used for the experimental group in the study. It is intended to reflect the theory of instruction proposed in the paper, providing instruction, practice, and feedback for each of the 4 skills incrementally.
Any errors or typos in it were not corrected because they were included in the study.

A Brief Introduction to Programming

Date: _____

Name: _____

Student ID: _____

Introduction

In this study, you will learn the basics of Python, one of the most popular programming languages in the world.

This packet will teach you programming by four distinct skills which are critical to programming:

1. **Reading code**: You will learn to read lines of code and predict how the code will execute.
2. **Writing code**: You will learn to write correct code that a computer would be able to run.
3. **Reading code templates**: *Templates* are common reusable patterns in code. You will learn to recognize several templates and their uses.
4. **Writing code with templates**: Given a problem description, you will learn to identify a template that solves this problem and write the code to do so.

Although most people think about programming as writing code to solve problems, reading code is also an important skill! In fact, software engineers spend most of their time reading code.

In about 2.5 hours of instruction and practice, you will be able to write simple Python programs! Let's get started!

Lesson structure

For this packet, we will present a new code construct and then provide you practice with it. We will show Python code in `monotype typeface`.

Note that some parts of this instruction are specific to the Python programming language and may be a bit different in other programming languages. We will explicitly say "in Python" when this occurs.

Initial as you go!

At the bottom of each page, we ask you to write your initials at the bottom of the page. This helps us understand how far along you got in the lesson. Please do that for this page and all following pages!

Initial here after reading page: _____

Teaching with metacognition

Metacognition is the awareness, understanding, and management of one's own thought processes. It is a critical part to learning. To help you make the most out of practice, we'll provide you with metacognitive prompts as you practice reading and writing code. So when you write code, we may ask you to plan out your code beforehand or write comments next to each line of code afterwards. Research suggests that metacognitive prompts like these will help you make the most out of your learning.

1) Planning your code

For some practice problems, we'll ask you to describe a step-by-step plan that describes the code you will write. This plan should be detailed enough for a stranger to look at it and follow along with your thought process when you are coding.

In example, say my task was to give change to a customer when they paid with a $20 bill for an item that costed $3.59.

My plan might look like this:

1. *Determine the total amount of money I need to give customer as change by subtracting the cost of the item from the amount they gave me.*
2. *Determine the number of $10 bills I need by subtracting 10 from the total amount until there is less than $10 left to give customer.*
3. *Repeat step 2 for $5, then $1*
4. *When less than $1 remains, I repeat step 2 with quarters, then dimes, then nickels, then pennies.*
5. *I stop whenever the remaining amount to give to the customer is 0.*

To summarize, please put effort into the metacognitive prompts (comment your code, plan your code) as well as reading and writing code. The prompting and practice will help you better learn how to code!

2) Commenting your code

A common thing that programmers do is to add *comments* to their code. While comments do not actually add functionality to the code, they do enable other people to better understand what the code is doing. In Python, comments exist to the right of **#** (a hashtag/pound sign) and extend to the end of the line.

In the example below, the 3 comments are highlighted. We see that highlights can appear on their own line (first comment, third comment), or sharing a line with some code that the computer will actually run (second comment).

```
# this is the first comment

spam = 1  # and this is the second comment
          # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

Initial here after reading page: _____

While the comments in the example below are a bit meaningless, we'll ask you to write meaningful comments on your code when you write code. Remember: comments exist on the right side of a `#`.

How Code Runs

Computers typically run code one line at a time from top to bottom, left to right. When reading and writing code, you too should run code from top to bottom, left to right.

When solving problems that ask you to read code, you will want to "be the computer." You can do that by following the steps below:

1. **Read the question.** Understand what you are being asked to do.
2. **Find where the code begins executing**. This is often the first line.
3. **Run the code one line at a time.**
    a. From the code, determine the rule for each part of the line (you're going to learn these rules!)
    b. Follow the rules
    c. Find the code for the next part.
    d. Repeat until the program terminates.

One step of this strategy has to do with keeping track of stored values (*variables*). We'll come back to that when we teach variables.

Initial here after reading page: _____

Overview

That is an overview of what we will cover. This will all make sense by the end of the instruction!

Concept	Description	Examples
Data Types	Different classifications of data (string, boolean, integer, float)	`"Hello"` `True` `3` `3.1`
Variables	Store values to be used later. Can be updated.	`cost = 1.50` `cost = 1`
Arithmetic operations	Math operations to be done between numbers	`8 / 2` `(3 + 1) * 4` `7 % 3`
Print statements	Output values to be displayed on the console	`print("hello world!")`
Relational operations	Determine if a relationship between two values is valid or not	`3 < 7` `"hi" == "HI"`
Conditionals	Execute different code based on condition	`cost = 1.4` `if (cost < 1):` `  print("buy it!")` `else:` `  print("do not buy")`

Initial here after reading page: _____

Reading Data Types

Computers reason about different data types so they can be precise about their reasoning. This is the same kind of precision as in Math: adding two apples and three dogs wouldn't make sense. By having different data types, Python can help you avoid mistakes like this!

Types are different classifications for data. In Python, 3 common types of data are numbers, strings, and boolean. We show examples of each data type below.

Data Type	**Example**	**Description**
integer (number)	`1, 2018`	Number that does not have a decimal point
float (number)	`1.0, 3.1415`	Number with a decimal point
string	`"Hello", 'hello', "123", "%+1+2"`	A characters surrounded by quotes ('single' or "double")
boolean	`True, False`	Truth values (can only be `True` or `False`)

Numbers can be whole numbers or decimals. Whole numbers are known as **integers** (e.g. `1, 2, 3`) and decimal numbers are known as **floats** (e.g. `3.14, 1.0)`. Note that floats have a decimal point; integers do not. So `1.0` and `1.` are both floats but `1` (no decimal point) is an integer.

Strings are sequences of characters enclosed in 'single quotes' or "double quotes". They are treated as *literals* where the characters inside the quotes are typically not executed and will appear "literally" as they are.

Boolean are truth values for logic. They can only be `True` or `False`. Note that the first letter is uppercase. `true` or `false` (lowercase) would cause your code to fail and not run.

Now let's get some practice with some of Python's data types!

Initial here after reading page: _____

Practice Reading Data Types

Circle the data type of each value:

	Value	Type
1	`123`	integer / float / string / boolean
2	`"123"`	integer / float / string / boolean
3	`123.0`	integer / float / string / boolean
4	`True`	integer / float / string / boolean
5	`'False'`	integer / float / string / boolean
6	`"123.0 False"`	integer / float / string / boolean

Initial here after reading page: _____

Practice Reading Data Types **Solutions**

Check your solutions for the following practice problem:

Circle the data type of each value:

	Value	Type
1	`123`	**integer** / float / string / boolean
2	`"123"`	integer / float / **string** / boolean
3	`123.0`	integer / **float** / string / boolean
4	`True`	integer / float / string / **boolean**
5	`'False'`	integer / float / **string** / boolean
6	`"123.0 False"`	integer / float / **string** / boolean

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Writing Data Types

Now let's talk about some things to remember when writing code that has data types:

- Strings always have quotes around them. In Python, these quotes can be 'single' or "double" quotes. Be sure strings are wrapped in the same quotes though! So `"hello"` and `'hello'` are both valid strings, but `'hello"` (with different quotes on each side) would result in an error.
- Numbers are floats if they have a decimal point (`1.1`, `1.0`, `1.`) and integers if they do not (`1`, `2`, `3`).
- Boolean are either `True` or `False`. They do not have quotes around them, otherwise they would be Strings. Remember to capitalize the first letter!

Now let's get some practice writing data types!

Initial here after reading page: _____

Writing Data Types Practice

1) Write 324 as an integer, float, and string:

	Value	Type
1		Number (integer)
2		Number (float)
3		String

---

2) There are only two Boolean values. Write them. Then write them as strings.

	Value	Type
1		Boolean
2		Boolean
3		String
4		String

Initial here after reading page: _____

Writing Data Types **Solutions**

Check your solutions for the following practice problem:

Write 324 as an integer, float, and string:

	Value	Type
1	`324`	Number (integer)
2	`324.0`	Number (float)
3	`"324"`	String

Write the 2 Boolean values. Then write them as strings.

	Value	Type
1	`True`	Boolean
2	`False`	Boolean
3	`"True"`	String
4	`"False"`	String

Remember: The first letter to a Boolean value is uppercase!

Check the boxes that apply:

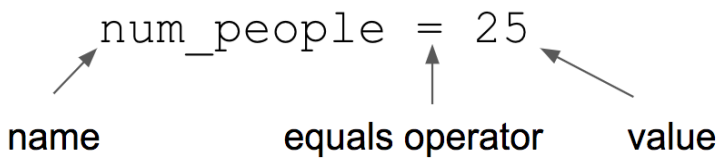
- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Reading a Variable Declaration

We often want to store values of various data types and use them later. We do this using variables.

For example, say we wanted to keep track of the number of people that are in a party. We could make or declare a variable to do this:



To declare this variable, we need a variable name on the left side (`num_people` in this case), an equal operator (a single equals sign: `=`) in the middle, and a variable value (`25` in this case) on the right side. This stores the value of the variable so we can reference it later using the variable name.

The name of the variable (left of the equals sign) is something you as the programmer get to think up, but there are a few rules as to what can go in a variable name in Python:

Rules for Variable Names	Bad Examples (not allowed in Python)	Good Examples (allowed in Python)
Variable names can only have letters, numbers, and underscores. Special characters other than underscore are not allowed in variable names (e.g. %, -, !, @, #, etc.)	`greeting%message` `excited_response!` `phone#`	`greeting_message` `excited_response` `phone_num`
Variable names cannot start with numbers.	`1st_period`	`first_period`
Some words in Python are special and variable names cannot be the same as them. In example, True and False would not be valid variable names because they are boolean values and the computer would get confused!	`True` `False`	`is_true` `is_false`
Variable names are **case-sensitive** (so my_var and MY_VAR would be names for different variables)	The following variable names are all different: `my_var` `My_Var` `MY_VAR`	

The value of a variable (right of the equals sign) is an integer in the example above but it can also be a string or Boolean value. It can even be another variable (more on that even later)!

Here are some example of code to declare variables:

Initial here after reading page: _____

Code to declare variables	Explanation
`first_name = "timmy"`	A variable named "first_name" is declared with a string value of "timmy"
`is_happy = True`	A variable named "is_happy" is declared with a Boolean value of True.
`age = 18`	A variable named "age" is declared with an integer value of 18.
`gpa = 3.5`	A variable named "gpa" is declared with a float value of 3.5.

Let's get some practice determining if variable names are valid!

Reading a Variable Declaration Practice

Determine if the given variable names are allowed in Python? If a variable name is not valid, briefly explain why not.

	Variable Name	Allowed in Python?	If not allowed, why not?
1	`my_variable`	Yes / No	
2	`1st_place`	Yes / No	
3	`_important_`	Yes / No	
4	`first-name`	Yes / No	
5	`True`	Yes / No	
6	`false`	Yes / No	

Initial here after reading page: _____

Reading a Variable Declaration Practice Solutions

Check your solutions for the following practice problem:

Determine if the given variable names are allowed in Python? If a variable name is not valid, briefly explain why not.

	Variable Name	Allowed in Python?	Notes
1	`my_variable`	**Yes**	
2	`1st_place`	**No**	Variable names cannot start with a number
3	`_important_`	**Yes**	Underscores are the only special characters allowed in variable names.
4	`first-name`	**No**	Only letters, numbers, and underscores are allowed in variable names. Dashes are not.
5	`True`	**No**	`True` is a special word because it is a boolean value, so it can't be a variable name.
6	`false`	**Yes****	**This name is technically ok, but it is not recommended because you could easily confuse this with the boolean value `False`!

Great job determining if a variable name is valid in Python! Now let's get practice on determining if variable declarations are valid.

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Reading a Variable Declaration Practice

Determine if the following lines of Python code correctly declares a variables. If the code does not declare a variable, briefly explain why not.

	Code to Declare Variable	Allowed in Python?	If not allowed, why not?
1	`my_var = "two"`	Yes / No	
2	`"hello" = reply`	Yes / No	
3	`happy!response = "yay!"`	Yes / No	
4	`NAME = "abby"`	Yes / No	
5	`should_run == False`	Yes / No	

Initial here after reading page: _____

Reading a Variable Declaration Practice Solutions

Check your solutions for the following practice problem:

Determine if the lines of Python code properly declares a variables. If the code does not declare a variable, briefly explain why not.

	Variable Name	Allowed in Python?	If not allowed, why not?
1	`my_var = "two"`	Yes	
2	`"hello" = reply`	No	The variable name should be on the left of the equal operator and value on the right.
3	`happy!response = "yay!"`	No	Only letters, numbers, and underscores are allowed in variable names, not exclamation points.
4	`NAME = "abby"`	Yes	
5	`should_run == False`	No	To declare a variable, you use a single equal signs. The double equals sign (==) checks for equality, something we'll learn later.

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- CORRECT: I was able to get the *correct* answer(s) without looking at the solutions Yes: ☐ / No: ☐

Great job! Now sometimes we want to update the values we stored in variables. We'll cover how to do that next.

Initial here after reading page: _____

Completing the Strategy on Reading Code

Before we progress further in learning about reading variables, let's take a break to complete our strategy on reading code. Recall in the beginning of this instruction we provided you with a strategy for reading code. We noted that the strategy was missing a piece related to keeping track of variables. Let's complete the strategy now. The highlighted line is the missing part of the strategy.

When solving problems that ask you to read code, you will want to "be the computer." You can do that by following the steps below:

1. **Read the question.** Understand what you are being asked to do.
2. **Find where the code begins executing**. This is often the first line.
3. **Run the code one line at a time.**
    a. From the code, determine the rule for each part of the line (you're going to learn these rules!)
    b. Follow the rules
    c. Update the memory table(s).
    d. Find the code for the next part.
    e. Repeat until the program terminates.

Memory tables are representations of the computer memory that stores variables and their values. An example is provided below:

Name	Value

To use a memory table:

1. When a variable is created, add it as a row in the table (variable name in the "Name" column; variable value in the "value" column).
2. When a variable is updated (which you'll learn next), find the variable by name, cross out the previous value and write in the new one.

So when a variable is created, a new row is added to the table. When a variable is updated, the previous value (in the value column) is crossed out and a new one is written down. We'll see an example of this on the next page.

Initial here after reading page: _____

Reading a Variable Update

After we have declared a variable, we can ***update*** the value of that variable to another value. Here's an example:

```
x = 1
y = 2
x = y
y = 3
```

In the first 2 lines, we declare a variable with variable name `x` and set it to an integer `1` and another variable `y` and set it to `2`. Let's show that in a memory table of variable names and values:

Variable Name	Value
x	1
y	2

In the 3rd line, we update variable `x` to also be the value of `y`:

Variable Name	Value
x	~~1~~ → **2**
y	2

The variable `x` has "erased" its previously value of `1` and now stores the value of `2`.

In the 4th line, we update variable `y` to also be 3.

Variable Name	Value
x	~~1~~ → 2
y	~~2~~ → **3**

Let's get some practice updating variables.

Initial here after reading page: _____

Reading Variable Updates Practice

Given the code below, do the following:

1. Put a checkmark (✓) to the left of each line that updates a variable.
2. Go through the code **line-by-line** and update the table to reflect each line of code running.
    a. For a variable declaration, write the new variable and value in an empty row
    b. For a variable update, find cross out the previous value and write in

```
___ num1 = 2

___ num2 = 1

___ temp = num1

___ num1 = num2

___ num2 = temp
```

Variable Name	Variable Value
`num1`	2

Initial here after reading page: _____

Reading Variable Updates Practice Solutions

Now check your work for the following practice problem:

Given the code below, do the following:

1. *Put a checkmark (✓) to the left of each line that declares a variable.*
2. *Go through the code **line-by-line** and update the table to reflect each line of code running.*
    a. *For a variable declaration, write the new variable and value in an empty row*
    b. *For a variable update, find cross out the previous value and write in*

_ `num1 = 2`

_ `num2 = 1`

_ `temp = num1`

✓ `num1 = num2`

✓ `num2 = temp`

Variable Name	Variable Value
`num1`	~~2~~ → 1
`num2`	~~1~~ → 2
`temp`	2

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- CORRECT: I was able to get the *correct* answer(s) without looking at the solutions Yes: ☐ / No: ☐

Fun fact: These lines of code form a "swap" operation where 2 variables (`num1`, `num2`) swap values by using a 3rd variable (`temp`) to temporarily store a value. This is an example of a reusable code template you'll learn to use later

Great job learning to *read* variable updates and declarations. Now let's move on to learning to *write* variable declarations and updates!

Initial here after reading page: _____

Writing Variable Declarations

When writing variable name declarations, remember that they should follow a similar structure:

- The left side has a variable name (made up of characters, numbers, and underscores)
- The middle has a single equals sign
- The right side has a variable value (e.g. a string, number, boolean, or name of another variable)

Writing Variable Updates

When updating a variable, the most important thing to do is make sure the variable you want to update has already been declared and you are referencing the right variable! A common yet dangerous mistake is updating the wrong variable because your code will still run even though you made a mistake.

An example of a variable update:

```
amount_to_pay = 0.00
drink = ""

drink = "soda"
amount_to_pay = 1.00
drink = "juice"
amount_to_pay = 1.50
```

In this example, we declare the variables `amount_to_pay` and `drink`. The `amount_to_pay` variable updates as we change `drink`, our drink choice.

Variables Changing Data Types

It is typically a good idea to have a variable to store the same data type even after updating it. This way it is easier to keep track of variables.

So if you declared a variable to be a string, it should stay a string.
If you declared a variable to be a boolean, it should stay a boolean.
If you declared a variable to be a number, it should stay a number.

Initial here after reading page: _____

Writing Variable Declarations Practice

In the box below, **write code that does the following:**

1. Declare a variable named `cost` and set it to `1.50`.
2. Declare a variable named `item` and set it to "`drink`".
3. Declare a variable `should_buy` and set it to `False`.
4. Update variable `cost` to `1.00`.
5. Update `should_buy` and set it to `True`.

After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Initial here after reading page: _____

Writing Variable Declarations Practice

Solutions

Now check your work for the following practice problem:

In the box below, write code that does the following:

1. Declare a variable named `cost` and set it to `1.50`.
2. Declare a variable named `item` and set it to "`drink`".
3. Declare a variable `should_buy` and set it to `False`.
4. Update variable `cost` to `1.00`.
5. Update `should_buy` and set it to `True`.

```
cost = 1.50 # declares variable and sets it to float
item = "drink" # declares variable and sets it to string
should_buy = False # declares variable and sets it to boolean

cost = 1.00 # updates variable
should_buy = True # updates variable
```

Notice how each line of the code has a comment next to it explaining what it does!

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- CORRECT: I was able to get the *correct* answer(s) without looking at the solutions Yes: ☐ / No: ☐

Initial here after reading page: _____

Template: Reading Variable Swaps

"You've learned how to read and write values of different data types and variable declarations. What can we do with this knowledge?" Let's introduce the notion of **variable swaps** with an example:

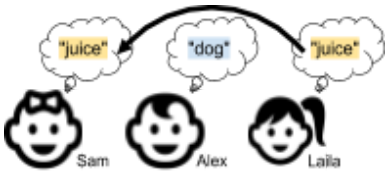
Let's say we have 2 babies, Sam and Alex. Each baby can only think of a single word at a time.

Sam and Alex want to "trade words" such that they end up thinking of the other baby's word. But they can only remember 1 word at a time, so how can they trade, or *swap*, thoughts? To do this, they can get the help of another baby, Laila!

This new baby can temporarily store a word while the first 2 babies swap thoughts! So one of the first 2 babies (say Alex) shares their thoughts with Laila:

Now that that Laila is storing Alex's original thought, Alex doesn't have to worry about remembering it. Sam can now share their thoughts with Alex:

So now that Alex has Sam's thought! Now Sam just need need's Alex's original thought. Good thing Laila remembers it! Laila can share Alex's original thought ("juice") with Sam.



Thanks to Laila's help, Sam and Alex were able to swap thoughts! If we imagine that each baby was a variable, we can apply the same logic to variable swaps.

Initial here after reading page: _____

A common task we want to do is swap the values in two variables so the result is that each variable stores the original value of the other variable. Because code runs one line at a time, there's no way to simultaneously swap variables. So just as Sam and Alex need the help of a 3rd baby, Laila, we need to use a 3rd temporary variable to store a value during the swap.

So a swap operation has 2 components:

- 2 declared variables which will update
- 1 temporary variable which will be declared and store another variable's value

Let's go through an example of swapping variables:

Here's an example of a swap where we swap the names of the winner and loser:

```
winner = "Abby"
loser = "Julian"
prev_winner = winner
winner = loser
loser = prev_winner
```

In the first 2 lines, we declare two variables:

Variable Name	Value
winner	"Abby"
loser	"Julian"

In the 3rd line, we declare a temporary variable (`prev_winner`) and set it so it has the same value as one of the two declared variables (winner in this case).

Variable Name	Value
winner	"Abby"
loser	"Julian"
prev_winner	**"Abby"**

Now that we stored the previous value of winner, go to the 4th line where we update winner with the new value (current value of loser)

Variable Name	Value
winner	~~**"Abby"**~~ → **"Julian"**
loser	"Julian"
prev_winner	"Abby"

In the last line, we update the variable loser with the original value of winner which is stored in our temporary variable (`prev_winner`).

Variable Name	Value
winner	~~"Abby"~~ → "Julian"
loser	~~**"Julian"**~~ → **"Abby"**
prev_winner	"Abby"

Let's practice recognizing swaps!

Initial here after reading page: _____

Practice Reading Variable Swaps

For each chunk of code, read the code and update the variable table. Write the final output in the "output" box, or write "error" if the output cannot be determined. Then, determine if float equality is correctly checked. If it is not, identify the error from the multiple choice options.

1)

```
player1 = "Jose"
player2 = "Kim"
temp = player1
player2 = player1
player1 = temp
```

Variable Name	Variable Value

Select one:

(a) There are not 2 values to be swapped.
(b) A temporary variable was not created.
(c) A temporary variable was created, but the variable update was not done correctly.
(d) This is a correct variable swap.

2)

```
num1 = 23
temp_val = num1
num2 = num1
num1 = temp
```

Variable Name	Variable Value

Select one:

(a) There are not 2 values to be swapped.
(b) A temporary variable was not created.
(c) A temporary variable was created, but the variable update was not done correctly.
(d) This is a correct variable swap.

3)

```
current_player = "Wanda"
next_player = "Jimmy"
current_player = next_player
next_player = current_player
```

Variable Name	Variable Value

Select one:

(a) There are not 2 values to be swapped.
(b) A temporary variable was not created.
(c) A temporary variable was created, but the variable update was not done correctly.
(d) This is a correct variable swap.

Initial here after reading page: _____

Practice Reading Variable Swaps Solutions

Check your work for the following problem:
For each chunk of code, determine if the variable swap is correct. If it is not, identify the error from the multiple choice options. We provide tables for you to track the variable updates as each line of code executes.

1)

```
player1 = "Jose"
player2 = "Kim"
temp = player1
player2 = player1
player1 = temp
```

Variable Name	Variable Value
player1	~~"Jose"~~ → "Jose"
player2	~~"Kim"~~ → "Jose"
temp	"Jose"

Select one:
(c) A temporary variable was created, but the variable update was not done correctly.

2)

```
num1 = 23
temp_val = num1
num2 = num1
num1 = temp
```

Variable Name	Variable Value
num1	~~23~~ → 23
temp_val	23
num2	23

Select one:
(a) There are not 2 values to be swapped.

3)

```
current_player = "Wanda"
next_player = "Jimmy"
current_player = next_player
next_player = current_player
```

Variable Name	Variable Value
current_player	~~"Wanda"~~ → "Jimmy"
next_player	~~"Jimmy"~~ → "Jimmy"

Select one:
(b) A temporary variable was not created.

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Template: Writing Variable Swaps

Now that we know how to read and recognize variable swaps, let's practice writing them!

To reiterate, the objective of a swap is to have two variables "trade" or "swap" values.

The most important thing about writing the code for a variable swap is making sure we swap the variables in the right order. If we swap out of order, we end up with the two variables equaling the same thing! To do a proper variable swap:

1. We must have two declared "permanent" variables whose values we will swap. Let's call them `var1` and `var2`.
2. We set a temporary variable to have the same value as one of the other variables. Let's call this variable `temp` and set it to have the same value as `var2`.
3. Update the variable whose value we stored in the temporary variable. So update `var2` to have the same value as `var1`.
    a. Because `temp` is storing the original value of `var2`, it's ok for `var2` to no longer have that value stored.
4. Update the value of `var1` to have the same value as the one stored in `temp` (which is `var2`'s original value).

If this is confusing, review the baby example from Reading Variable Swaps (a few pages back). Relative to this example, Sam is `var1`, Alex is `var2`, and Laila is `temp`.

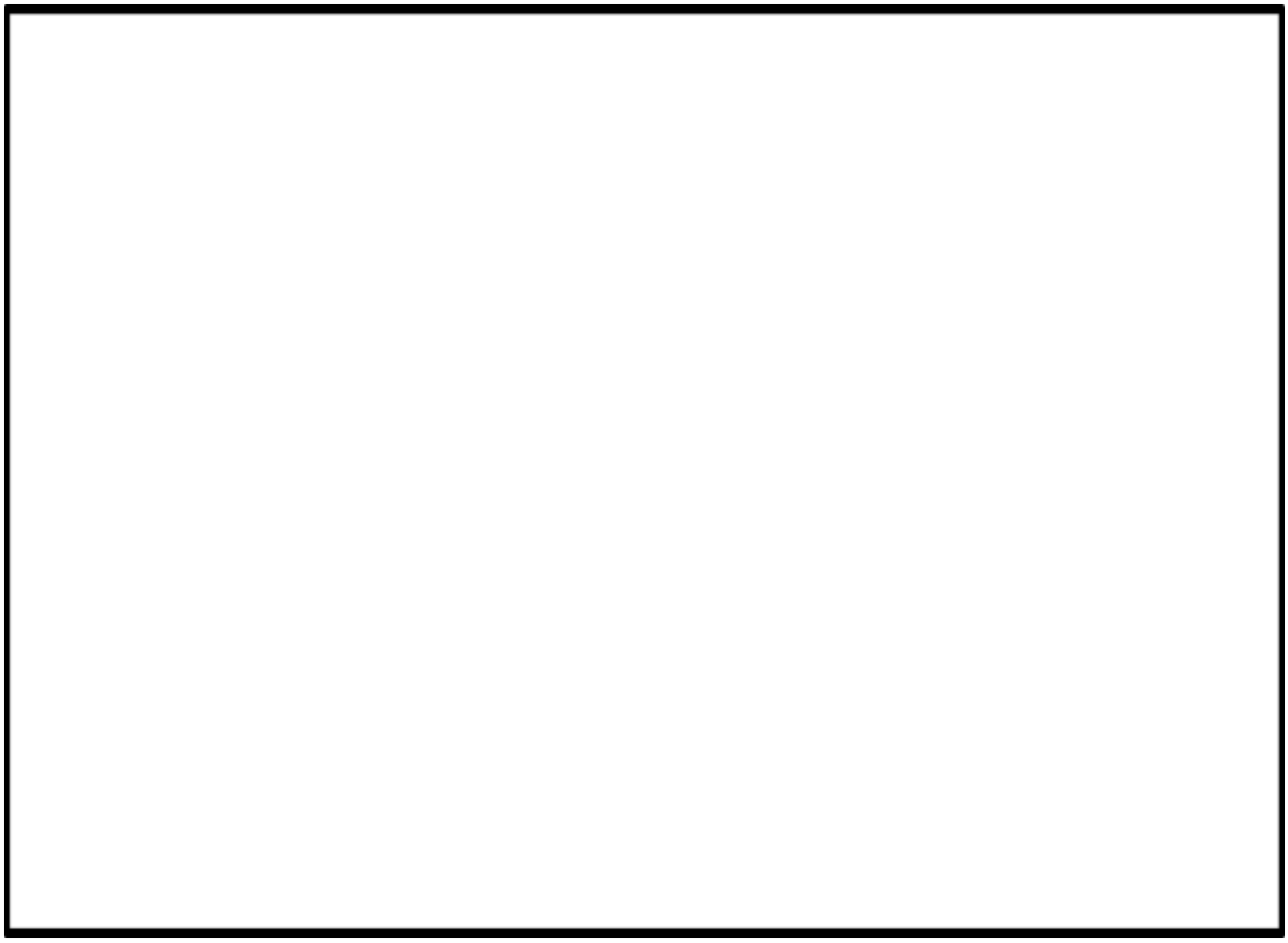
Let's get some practice writing a swap!

Initial here after reading page: _____

Template: Writing Variable Swaps Practice

Joyce is writing code to keep track of how much money she has paid towards a loan. She realizes she made an error and mixed up the values for the variables `amt_paid` and `amt_owed`. In reality, she has actually almost paid off the loan. Help Joyce by swapping the values so that the amount she has paid towards the loan (`amt_paid`) is greater than the amount owed (`amt_owed`)!

1) In plain English, describe a step-by-step plan for solving the problem:



Initial here after reading page: _____

2) Write the code to switch the values stored in `amt_paid` and `amt_owed`.

```
amt_paid = 231.89
amt_owed = 12152.23
```

If you want to check your work, read through the code line-by-line and fill out the variable table below:

Variable Name	Variable Value

3) After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Template: Writing Variable Swaps Practice Solutions

Check your work for the following practice problem:

Joyce is writing code to keep track of how much money she has paid towards a loan. She realizes she made an error and mixed up the values for the variables `amt_paid` *and* `amt_owed`*. In reality, she has actually almost paid off the loan. Help Joyce by swapping the values so that the amount she has paid towards the loan (*`amt_paid`*) is greater than the amount owed (*`amt_owed`*)!*

Plan:

```
Declare temporary variable and set it to value of 1 of variables.
Update the variable (whose value is stored in the temporary variable)
with the value in the other variable.
Update the other variable with the value in the temporary variable.
```

Code:

```
amt_paid = 231.89
amt_owed = 12152.23

temp = amt_paid # declare variable for swap
amt_paid = amt_owed # update variable with value of other variable
amt_owed = temp # update other variable in swap
```

Just to be sure we did this right, let's make a table to keep track of variable updates

Variable Name	Variable Value
amt_paid	~~231.89~~ → 12152.23
amt_owed	~~12152.23~~ → 231.89
temp	231.89

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Looks good! Now let's move on arithmetic operators so we can use code to make math calculations on numbers.

Initial here after reading page: _____

Reading Arithmetic Operators

Much of the things we experience on computers (watching movies, using spreadsheets, etc.) involve arithmetic with numbers (integers and floats). Python offers arithmetic operators to help implement these behaviors. These operations should be familiar if you remember what you learned from math class, but there are some important but subtle differences in how Python handles math.

Here are the common arithmetic operators in Python:

Operator	Explanation	Example
`+` Addition	Adds values on either side of operator	`30 + 21` (result is 51) `4.1 + -1.0` (result is 3.1)
`-` Subtraction	Subtracts value on right side from value on left side of operator	`5 - 2` (result is 3) `3.5 - 1.0` (result is 2.5)
`*` Multiplication	Multiples values on either side of operator	`3 * 2` (result is 6) `2.1 * 3` (result is 6.2)
`/` Division	Divides value on left by value on right	`8 / 2` (result is 4) `4.4 / 2.0` (result is 2.2)
`%` Modulus	Divides value on left by value on right and returns remainder	`4 % 2` (result is 0) `5 % 3` (result is 2)

The most unfamiliar arithmetic operator is likely the modulus (`%`) operator, which we use to determine the remainder of a division operation.

Determining which operations should be executed first is an important part of arithmetic operators. In math class you may have learned PEMDAS: Parentheses, exponents, multiplication, division, addition, and subtraction. Thankfully, this is still true in programming. The modulus operation (`%`) is in the same rank or order as multiplication and division. So, we calculate parenthesis, then exponents, then multiplication, division, and modulus, and then addition and subtraction.

In example, `5 % 2 + 1` results in 2 because modulus is a division, so we first calculate it (`5 % 2` results in 1) and then we calculate the addition operation (1 + 1).

So far, things should be pretty familiar. But recall that Python has two types for numbers: integers (numbers without a decimal point) and floats (numbers with decimal points). We should remember two things:

1. Integers drop the decimal value; they do not round.
2. Calculations involving a float always result in a float.

Let's go into a little more detail about these two important points:

Initial here after reading page: _____

Integers don't round!

What is 5 divided by 3? You could say 1 with a remainder 2. Or maybe you learned in math class to round the answer up to 2. But if you type `5 / 3` into Python, you actually get the integer `1` as a result. This is because **integers in Python do not round. Instead, they drop the decimal value**. This is equivalent to "rounding down" or taking the "floor" of the result. Python always drops any decimal value when you are making calculations between two integers.

Type Coercion: When integers become floats

Sometime we will do calculations that involve both integers and floats. In this case, the result will always be a float. So, `1.0 * 2` would result in `2.0`.

One last thing to note is that we can use variables that store numbers in our calculations. Here's an example:

```
cost_juice = 1.50
cost_juice * 10
```

The result of this code example is `15.0`.

Reading Arithmetic Operators Practice

For the following lines of code, determine the output type and the actual result

Code	Data Type (circle one)	Result
`5.0 + 2`	integer / float	
`9 / 2`	integer / float	
`my_value = 4` `my_value * 2`	integer / float	
`(4 + 2) / 2.0`	integer / float	
`7 % 2 + 1.1`	integer / float	

Initial here after reading page: _____

Reading Arithmetic Operators Practice Solutions

Check your solutions for the following practice problem:

For the following lines of code, determine the output type and the actual result

Code	Data Type (circle one)	Result	Explanation
`5.0 + 2`	integer / **float**	7.0	Any operation that involves a float will result in a float as output.
`9 / 2`	**integer** / float	4	Integers drop the decimal value. They do not round.
`my_value = 4` `my_value * 2`	**integer** / float	8	If a variable value is a number, we can use it in arithmetic operations.
`(4 + 2) / 2.0`	integer / **float**	3.0	
`7 % 2 + 1.1`	integer / **float**	2.1	Calculate the modulus before the addition operator.

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Writing Arithmetic Operators

When writing code with arithmetic operators, it is important to make sure to keep track of the data types of the values you are working with. The improper data types may result in your code not running or your code running and outputting unusual results! Here are a few things to keep in mind:

1) **Arithmetic operations can only be done on numbers (integers and floats).** We cannot perform arithmetic operations on any data type besides numbers. So `3 + "4"` would cause the code to fail because "4" is not a number (the quotes make it a string).

2) **Arithmetic operations on floats and integers behave differently.** If the calculations involve just integers, the output will be an integer. Any calculations involving floats will output a float value. This is important because integers do not round!

Initial here after reading page: _____

Writing Arithmetic Operators Practice

Write code that does the following:

- Declare a variable `val` and set it to `7` modulus `2`.
- Update the value of `val` by multiplying the current value by the sum of `1.0` and `0.2`.

After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Complete these statements about the code you wrote above:

- When the variable `val` is declared, it is a/an **integer / float** (circle one).
- When the variable `val` is updated, it is a/an **integer / float** (circle one).

Initial here after reading page: _____

Writing Arithmetic Operators Practice
Solutions

Check your answers for the following problem:

Write code that does the following:

- Declare a variable `val` and set it to `7` modulus `2`.
- Update the value of `val` by multiplying the current value by the sum of `1.0` and `0.2`.

```
val = 7 % 2 # set variable to result of 7 % 2
val = val * (1.0 + 0.2) # update variable by multiplying current value
```

Complete these statements about the code you wrote above:

- When the variable val is declared, it is a/an **integer** / variable (circle one).
- When the variable val is updated, it is a/an integer / **float** (circle one).

Explanation: In the first line, we use the modulus operator to get the remainder between 7 and 2, which is an integer because the arithmetic operation was between 2 integers. We then multiply the result by the sum of 2 floats (`1.0` and `0.2`). We include the parenthesis around these floats to ensure the values are summed before multiplying with the variable `val`. The output is a float because the arithmetic operation included a float.

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Now that we know how to read and write code with arithmetic operators, let's see how we can use them to read in each digit of a number!

Initial here after reading page: _____

Reading Arithmetic Operators Template: Processing Digits

These days, we have use numeric passwords to secure many things, such as cell phones, debit cards, doorways, and user accounts. Verifying that a numeric password, such as the 4 digit personal identification number (PIN) of a debit card, is correct is critical to ensuring security.

One of the simplest ways to verify if a number is valid was to check for an underlying relationship between the digits in a number. A very simple example is saying that a 4 digit PIN number is valid if the 2nd digit of a 4 digit PIN is larger than the 3rd digit. To do this, we have to be able to extract individual digits from the PIN number, which we can think of as an integer with multiple digits. In this section, we'll teach you a way to extract individual digits from an integer with multiple digits.

If we have an integer value with multiple digits (e.g. `23`, `1234`), we can use arithmetic operators to look at the integer 1 digit at a time. To do this we have to remember that integers don't round; they drop any decimal values when multiplying or dividing. By knowing this and using the modulus operator, we can repeatedly look at the rightmost digit, then remove it from the original value and then look at rightmost digit again.

Here are the steps for processing digits:

1. Start with an integer as input:
2. Repeat the following steps until every digit has been processed:
    a. Use the modulus operator to get the last digit and store that value.
    b. Use the division operator to remove the last digit

Let's go over an example on the next page!

Initial here after reading page: _____

Here's an example where we store every digit in the 3 digit value stored in the variable `input`:

```
input = 123

last_digit = input % 10
input = input / 10

second_digit =  input % 10
input = input / 10

first_digit = input % 10
```

The first chunk of code defines the input variable:

Variable Name	Value
input	123

The next chunk of code uses the modulus operator to get the last digit and stores it in the variable `last_digit`. It then updates the variable `input` by removing the last digit by dividing by 10.

Variable Name	Value
input	~~123~~ → 12
last_digit	3

The next chunk of code repeats the process as the previous chunk of code, using the modulus operator to get the next digit and then division to remove that value from the input

Variable Name	Value
input	~~123~~ → ~~12~~ → 1
last_digit	3
second_digit	2

In the last line of code, we store the first digit. We don't need to update input again because there are no more digits to store.

Variable Name	Value
input	~~123~~ → ~~12~~ → 1
last_digit	3
second_digit	2
first_digit	1

Let's get some practice reading code that processes digits!

Initial here after reading page: _____

Practice Reading Digit Processing

For each chunk of code, read the code and update the variable table. Then, determine if float equality is correctly checked. If it is not, identify the error from the multiple choice options.

1)

```
inp = 31
last_digit = inp % 10
inp = inp / 10
first_digit = inp % 10
```

Variable Name	Variable Value

Select one:

(a) The starting value is not an integer.
(b) Digits are not properly extracted from starting value.
(c) The starting value is not updated properly.
(d) This code processes digits correctly.

2)

```
start = 198
last_digit = inp % 10
inp = inp / 10
second_digit = inp % 10
inp = inp % 10
first_digit = inp % 10
```

Variable Name	Variable Value

Select one:

(a) The starting value is not an integer.
(b) Digits are not properly extracted from starting value.
(c) The starting value is not updated properly.
(d) This code processes digits correctly.

3)

```
current = 510
last = current % 10
current = current / 10
middle = current % 10
first = current % 10
current = current / 10
```

Variable Name	Variable Value

Select one:

(a) The starting value is not an integer.
(b) Digits are not properly extracted from starting value.
(c) The starting value is not updated properly.
(d) This code processes digits correctly.

Initial here after reading page: _____

Practice Reading Digit Processing Solutions

Check your work for the following practice problem:
For each chunk of code, read the code and update the variable table. Then, determine if float equality is correctly checked. If it is not, identify the error from the multiple choice options.

1)

```
inp = 31
last_digit = inp % 10
inp = inp / 10
first_digit = inp % 10
```

Variable Name	Variable Value
inp	~~31~~ → 3
last_digit	1
first_digit	3

Select one:
(d) <u>This code processes digits correctly.</u>

2)

```
start = 198
last_digit = inp % 10
inp = inp / 10
second_digit = inp % 10
inp = inp % 10
first_digit = inp % 10
```

Variable Name	Variable Value
start	~~198~~ → ~~19~~ → 9
last_digit	8
second_digit	9
first_digit	9

Select one:
(b) <u>Digits are not properly extracted from starting value.</u>

3)

```
current = 510
last = current % 10
current = current / 10
middle = current % 10
first = current % 10
current = current / 10
```

Variable Name	Variable Value
current	~~510~~ → ~~51~~ → 5
last	0
middle	1
first	1

Select one:
(c) <u>The starting value is not updated properly.</u>

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Writing Arithmetic Operators Template: Processing Digits

We want to use the processing digits template anytime we want to access a specific digit in an integer with multiple digits.

To process digits:

1. Ensure the input is an integer
2. For each digit in the input:
    a. Extract the rightmost digit by taking the modulus 10 (`% 10`) of the input and storing it in a new variable.
    b. Remove the rightmost digit from the input by dividing it by 10 and updating the variable.

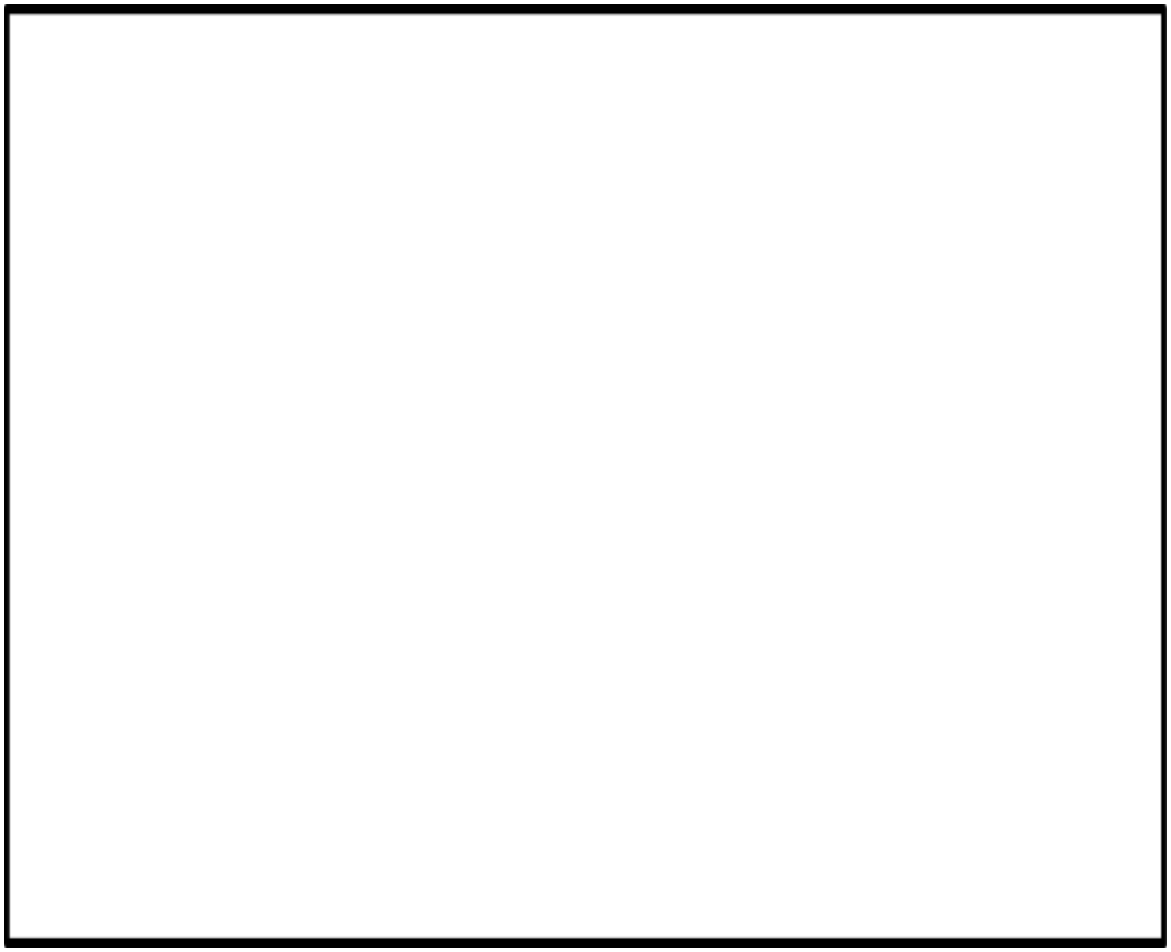
Remember to only update the input value and remove the rightmost digit after you have stored that rightmost value, otherwise it is lost for good!

Let's practice writing code that uses the processing digits template:

Template: Writing Digit Processing Practice

Given 2 integers as input (stored in `input_a` and `input_b`), write code that sums all the digits in the two inputs and stores the sum in a variable `total`.

1) In plain English, describe a step-by-step plan for solving the problem:



Initial here after reading page: _____

2) Write the code to sum all the digits in `input_a` and `input_b`.

```
input_a = 23
input_b = 314
```

Here's a table in case you want to keep track of your variables.

Variable Name	Variable Value

3) After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Template: Writing Digit Processing Practice Solutions

Check your work for the following practice problem: *Given 2 integers as input (stored in `input_a` and `input_b`), write code that sums all the digits in the two inputs and stores the sum in a variable `total`.*

Plan:

1. Extract digits for input_a by using % 10 to get the digit and storing it in a variable
2. Remove the digit from input_a by dividing input_a by 10
3. Repeat steps 1 and 2 for all digits in input_a, and then for input_b
4. Sum variables that store digits of input_a and input_b

(This uses the digit processing template)

Code:

```
input_a = 23
input_b = 314

a_last = input_a % 10    # store last digit
input_a = input_a / 10   # remove last digit
a_first = input_a % 10   # store first digit

b_last = input_b % 10    # store last digit
input_b = input_b / 10   # remove last digit
b_middle = input_b % 10  # store middle digit
input_b = input_b / 10   # remove middle
b_first = input_b % 10   # store first digit

total = a_last + a_first + b_last + b_middle + b_first # sum all digits
```

Variable Name	Variable Value
input_a	~~23~~ → 2
input_b	~~314~~ → ~~31~~ → 3
a_last	3
a_first	2
b_last	4
b_middle	1
b_first	3
total	13

Just to be sure we did this right, let's make a table to keep track of variable updates

Let's take a moment to appreciate all this code that you have written! You declared and updated multiple variables to perform a complex task. Well done!

Let's move on to learning about print statements which we can use to output values.

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Reading Print Statements

Thus far you have learned about different data types that values might have, you have learned how to store values with variables, and you have learned to do mathematical operations on values. While the computer has all these values stored, sometimes we want to *see* the values outputted. We may want to do this to show values or outputs to ourselves or other people. To do so, we need to use ***print*** statements.

The print statement sends an output to your monitor. Don't let the name fool you, print statements usually have nothing to do with a printer!

A print statement is a *function* built-into Python. A function is a block of code that performs a single, reusable action. We can execute the reusable code in a function by *calling* it. We call a function by its name followed by parenthesis where anything in the parentheses would be input value that would be passed into a function's code.

An example of a print statement is `print("hello!")`. This would output "hello!", as shown in the image below of a *console*, a text entry display on a computer:

```
>>> print("hello!")
hello!
```

Here, ">>>" in the first line is not typed; it is just the console signaling that that line was typed. So, the programmer types `print("hello!")` in the top line and then the computer responds with the second line, "hello!".

Here are 3 rules to keep in mind about print statements:

Rule	Example
Operations (such as addition) in the input will execute before values are printed.	`>>> print(1+2+3)` `6`
If we pass in variables, we print the variables' values	`>>> cost = 1.50` `>>> print(cost)` `1.5`
Each print statement outputs on a new line.	Input: `print("hi")` `print("bye")` Output: `hi` `bye` *Note: ">>>" is missing from this example because we didn't write this code in a console. Don't worry about this for now. We'll cover how to run Python code on the computer at the end of this lesson!*

Initial here after reading page: _____

Reading Print Statements Practice

Determine the output of the print statements below:

	Code	Output
1	`print("hello world")`	
2	`print(5 - 2.0)`	
3	`val = 7` `print(val)` `print(val + 1)` `val = 3` `print(val)`	
4	`print("3+5")`	
5	`should_pay = False` `print(should_pay)`	

Initial here after reading page: _____

Reading Print Statements Practice Solutions

Determine the output of the print statements below:

Code	Output	Explanation
`print("hello world")`	**`hello world`**	
`print(5 - 2.0)`	**`3.0`**	Operations in the input (subtraction in this case) execute before printing.
`val = 7` `print(val)` `print(val + 1)` `val = 3` `print(val)`	**`7`** **`8`** **`3`**	Each print statement goes on its own line.
`print("3+5")`	**`3+5`**	Strings don't execute!
`should_pay = False` `print(should_pay)`	**`False`**	

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Great job! Now let's move on to writing print statements.

Initial here after reading page: _____

Writing Print Statements

When writing code to print values to the console, we use a print statement by typing the word "print" followed by what we want to print. We wrap what we want to print in parentheses.

When using a print statement, there a few things to remember to make sure you write correct code:

1. Wrap the value to print in parentheses
2. We can't store print values in a variable (that just wouldn't make sense!)

Here are a few examples of code with print statements that would NOT run

Bad Code (would not run)	Corrected Code	Explanation
`("hello")print`	`print("hello")`	The word "print" comes before the value to print
`print(1 + 3`	`print(1 + 3)`	Parentheses not closed
`x = print(1)`	`x = 1` `print(x)`	Cannot store a print statement in a variable

Let's get some practice writing print statements!

Initial here after reading page: _____

Writing Print Statements Practice

1) Write code that does the following:

- Declares a variable `inp` and set it to `5`.
- Print the string "begin"
- Print the value stored in `inp`.
- Print the value stored in `inp` multiplied by `2`.
- Print the value stored in `inp` modulus `3`.

2) After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

3) Write the output of this code below:

Initial here after reading page: _____

Writing Print Statements Practice

Solutions

Write code that does the following:

- Declares a variable `inp` and set it to `5`.
- Print the string "begin"
- Print the value stored in `inp`.
- Print the value stored in `inp` multiplied by 2.
- Print the value stored in `inp` modulus 3.

```
inp = 5             # set variable to number
print("begin")      # print string literal
print(inp)          # print value in inp
print(inp * 2)      # set value in inp multiplied by 2
print(inp % 3)      # print remainder of value in inp divided by 3
```

Write the output of this code below:

```
begin
5
10
2
```

Remember that print statements and variable updates are separate things. So even though we wrote code to do arithmetic operations on the variable `inp` and print the output, we never actually wrote code to update the variable. So the value stored in `inp` never changes!

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Reading Relational Operators

You now know about different types of data, how to do mathematical operations with them, and how to store and output the values. A common task we want to do with values is determine their relationship with each other. In example, we may only wear a jacket if the temperature is less than a certain threshold value. Or we may choose to bring an umbrella if it is raining. Or we may choose to let someone into your home only if they know the password. **Relational operators determine the relationship between values.**

Relational operators test to determine if a given relationship is valid. The result in a boolean value of `True` if the relationship is in fact valid and `False` if the relationship is not valid.

Common relational operators

Relational operator	Explanation	Example
== equals	Determine if 2 values are exactly equal	`1 == 1` (result would beTrue) `"A" == "A"` (result is True) `"A" == "a"` (result is False) `True == False` (result is False)
!= not equals	Determine if 2 values are not equal	`1 != 1` (result would be False) `"A" != "A"` (result is False) `"A" != "a"` (result is True) `True != False` (result is True)
< Less than	Determine if the left value is smaller than the right value	`1 < 1` (result: False) `-1 < 1` (result: True) `10 < 9` (result: False) `3.14 < 5` (result: True)
> Greater than	Determine if the left value is larger than the right value	`1 > 1` (result: False) `-1 > 1` (result: False) `10 > 9` (result: True) `3.14 > 5` (result: False)
<= Less than or equal to	Determine if the left value is smaller or equal to the right value	`1 <= 1` (result: Ture) `-1 <= 1` (result: True) `10 <= 9` (result: False) `3.14 <= 5` (result: True)
>= Greater than or equal to	Determine if the left value is larger or equal to the right value	`1 >= 1` (result: True) `-1 >= 1` (result: False) `10 >= 9` (result: True) `3.14 >= 5` (result: False)

Notice how string values are case-sensitive, so "a" and "A" are not equal.

Let's get some practice reading relational operators!

Initial here after reading page: _____

`and` operator: for chaining relational operators

Sometime we want to check multiple relationships at the same time. In example, we might only go for a walk if it is warmer than 50 degrees *and* cooler than 80 degrees. In Python, this compound relationship would look like this: `temp > 50 and temp < 80`.

The `and` operator only returns true if the relationships on both its left and right sides are true. Here are some examples:

Code	Output	Explanation
`temp = 60` `print(temp>50 and temp<80)`	**`True`**	The relationships on both sides of the `and` are True, so the entire relationship evaluates to True.
`temp = 90` `print(temp>50)` `print(temp<80)` `print(temp>50 and temp<80)`	**`True`** **`False`** **`False`**	Because temp is not < 80, the 3rd print statement evaluates to False because the relationship to the right of the `and` is False.
`x = 1` `y = 2` `z = 3` `print(x<y and x<z)` `print(y<x and y<z)`	**`True`** **`False`**	The first print statement is True for relationships on both sides of the `and`. The second print statement is False because the relationship on the left side of the `and` is False.

Initial here after reading page: _____

Reading Relational Operators Practice

Determine if the result of the following lines of code are True or False:

Code	Result
`1.2 < 5`	True / False
`x = 5` `y = 4` `(x + x) <= y and x <= (y+y)`	True / False
`"HELLO" == "hello"`	True / False
`a = 1` `b = 1` `a != b`	True / False
`3 >= (5 % 2)`	True / False
`hello = "Hello"` `hello == "Hello"`	True / False

Initial here after reading page: _____

Reading Relational Operators Practice
Solutions

Check your answers for the following question:

Determine the result of the following lines of code:

Code	Result	Explanation
`1.2 < 5`	**True**	
`x = 5` `y = 4` `(x + x) <= y and x <= (y+y)`	**False**	The computer calculates arithmetic operation (addition) first, so the resulting comparison is `10 <= 4`.
`"HELLO" == "hello"`	**False**	Strings are case-sensitive
`a = 1` `b = 1` `a != b`	**False**	
`3 >= (5 % 2)`	**True**	Again, the computer calculates the arithmetic operation (modulus) first, so the resulting comparison is `3 >= 1`
`hello = "Hello"` `hello == "Hello"`	**True**	This code is tricky because the first line sets a variable named `hello` to the string value "Hello". The second line checks for equality to determine if the value stored in the variable is equal to "Hello" (with an uppercase "H"). They are both "Hello", so the result is True.

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Writing Relational Operators

Similar to arithmetic operators, relational operators must sit between two values or variables. The values or variables you are comparing with a relational operator should typically be of the same type.

When writing code that uses relational operators, here are a few things to keep in mind:

1. **= is for variable assignment; == is for checking equality.** A single equals sign is for assigning a value to a variable (e.g. `x = 1` assigns the value `1` to variable `x`). A double equals sign is for checking for equality between two values (e.g. `x == 1` determines if the value stored in variable `x` is equal to `1`.)
2. **String equality is case-sensitive.** Strings are equal if they have the exact same characters. Upper and lowercase letters are seen as different characters, so strings must be the same case to be equal.
3. **It's best to have values be of the same type.** Comparing values of different type (comparing a string to an integer for example) rarely makes sense and may often result in unexpected outputs. This strange behavior is especially true for comparing integers and floats, as
4. **`and` can be used to check multiple relationships.** If we want to only return True if multiple relationships all evaluate to True, we can put the `and` operator between relationships.

Let's get some practice writing code with relational operators.

Initial here after reading page: _____

Writing Relational Operators Practice

Write code that does the following:

- Declare a variable `greet1` and set it to `"hello"`.
- Declare a variable `greet2` and set it to `"Hello"`.
- Declare a variable `diff_greet` and set it to the result of whether `greet1` and `greet2` are not equal.

After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Write code that does the following:

- Declare a variable `val1` and set it to `93`.
- Declare a variable `val2` and set it to the variable `val1`.
- Declare a variable `one_is_less` and set it to the result of whether `val1` is less than or equal to `val2`.
- Declare a variable `check_both` and set it to the result of `val1 % 2 == 0` and `one_is_less`

After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Initial here after reading page: _____

Writing Relational Operators Practice

Solutions

Check your solutions for the following questions:

Write code that does the following:

- *Declare a variable `greet1` and set it to `"hello"`.*
- *Declare a variable `greet2` and set it to `"Hello"`.*
- *Declare a variable `diff_greet` and set it to the result of whether `greet1` and `greet2` are not equal.*

```
greet1 = "hello"                # set variable to string literal
greet2 = "Hello"                # set variable to string literal
diff_greet = greet1 != greet2   # set variable to result of relational operation (boolean)
```

Write code that does the following:

- *Declare a variable `val1` and set it to `93`.*
- *Declare a variable `val2` and set it to the variable `val1`.*
- *Declare a variable `one_is_less` and set it to the result of whether `val1` is less than or equal to `val2`.*
- Declare a variable `check_both` and set it to the result of `val1 % 2 == 0` and `one_is_less`

```
val1 = 93                       # set variable to int
val2 = val1                     # set variable to value of other variable (int)
one_is_less = val1 <= val2      # set variable to result of relational operation (boolean)

check_both =  val1 % 2 == 0 and one_is_less # check if val1 is even and if one_is_less
is True and store result
```

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Reading Float Equality Template

Computers make very precise calculations. This often a good thing, because we want precision if we are calculating things such as a satellite's trajectory around the earth where small changes can result in major differences. This precision can often result in some *strange* behavior though. Let's see an example of that:

So does 1.0 equal to 1.000000000000001? If we were a cashier giving change, maybe we would think these were the same number. But our math teacher would probably tell us these numbers are not equal. Let's see what Python thinks:

```
>>> 1.0 == 1.000000000000001
False
```

Python agrees that these floats are not equal. But does 1.0 equal to 1.0000000000000001? (1 additional zero added). Our math teacher would probably still think they are not equal. Let's see if Python thinks differently:

```
>>> 1.0 == 1.0000000000000001
True
```

Interesting! Python believes that these values are the same. Why this is true is because of how the computer stores float values and an arbitrary cutoff that most (but not all) computers have to determine equality. This isn't ideal, so **let's *NOT* use the equal operator to check float equality**.

A better way to do float equality is saying that numbers that are "close enough" are equal where we define the threshold for close enough. We can do this by seeing if the distance or absolute difference between two floats is less than some threshold that *we* define.

Float equality determines if float values are close enough to be considered equal.

To determine float equality:

1. Define a threshold value (small positive float) and set it to a variable.
2. Calculate the absolute difference between two numbers (integers or floats) using the `abs()` function.
3. Determine if the absolute difference is less than the threshold value.

Let's see an example of calculating float equality:

Initial here after reading page: _____

In this example, we have 3 float values (stored in variables `f1, f2, f3`) and we want to determine if they are equal.

```
f1 = 1.111
f2 = 1.112
f3 = 1.1112
```

To do this, we'll first define a threshold. Floats that are less than this threshold distance apart are considered equal. We'll declare a variable `threshold` and set it is `0.001`.

```
f1 = 1.111
f2 = 1.112
f3 = 1.1112

threshold = 0.001
```

Now we'll check if the floats are equal. To do this, we check if the absolute distance between the floats are less than the threshold we defined. To get the absolute value, we can use `abs()` function which returns the absolute value of the input. So `abs(-1)` would be `1` and `abs(1.1)` would be `1.1`. So we take the absolute value of the difference of 2 floats and see if that is less than the threshold value. Let's check for equality between `f1` and `f2` and between `f1` and `f3` and print the outcome.

```
f1 = 1.111
f2 = 1.112
f3 = 1.1112

threshold = 0.001

print( abs(f2 - f1) < threshold )
print( abs(f3 - f1) < threshold )
```

The outcome:

```
False
True
```

So given the threshold we defined, `f1` and `f2` are not equal and `f1` and `f3` are equal.

Let's practice reading float equalities.

Practice: Reading Float Equality Template

For each chunk of code, read the code and update the variable table. Write the final output in the "output" box, or write "error" if the output cannot be determined. Then, determine if float equality is correctly checked. If it is not, identify the error from the multiple choice options.

1)

```
a = 1.21
b = "1.211"
e = 0.001
d = abs(a-b)
print(d < e)
```

Variable Name	Variable Value
Output:	

Select one:

(a) The values to compare are not numbers (integer or float)
(b) The threshold value is defined but is not a small positive number
(c) The absolute difference is not compared to be less than a threshold
(d) This code checks float equality correctly.

2)

```
x = 2
y = 2.000000004
xy = x-y
print( xy < 0.00001)
```

Variable Name	Variable Value
Output:	

Select one:

(a) The values to compare are not numbers (integer or float)
(b) The threshold value is defined but is not a small positive number
(c) The absolute difference is not compared to be less than a threshold
(d) This code checks float equality correctly.

3)

```
a = 1.0000002
b = 1.0000001
e = -0.000001
d = a - b
print(abs(d)<e)
```

Variable Name	Variable Value
Output:	

Select one:

(a) The values to compare are not numbers (integer or float)
(b) The threshold value is defined but is not a small positive number
(c) The absolute difference is not compared to be less than a threshold
(d) This code checks float equality correctly.

Initial here after reading page: _____

Practice: Reading Float Equality Template Solutions

Check your answer for the following questions: *For each chunk of code, determine if float equality is correctly checked. If it is not, identify the error from the multiple choice options.*

1)

```
a = 1.21
b = "1.211"
e = 0.001
print(abs(a-b)<e)
```

Variable Name	Variable Value
a	1.21
b	"1.211"
e	0.001
d	error
Output:	error

Select one:

(a) <u>The values to compare are not numbers (integer or float)</u>

2)

```
x = 2
y = 2.000000004
z = x - y
print( z < 0.00001)
```

Variable Name	Variable Value
x	2
y	2.000000004
z	-0.000000004
Output:	True

Select one:

(c) <u>The absolute difference is not compared to be less than a threshold</u>

3)

```
a = 1.0000002
b = 1.0000001
e = -0.000001
d = abs(a-b)
print(abs(a-b)<e)
```

Variable Name	Variable Value
a	1.0000002
b	1.0000001
e	-0.000001
d	0.0000001
Output:	False

Select one:

(b) <u>The threshold value is defined but is not a small positive number</u>

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Writing Float Equality Templates

We use the float equality template anytime we want to compare the relationship between numbers that could be floats. That means that one or both of these numbers could be an integer and this check of equality would still be valid. That is, `3 == 3` would be the same result as `abs(3 - 3) < 0.0001`.

The steps to check float equality and common mistakes for each step:

Step	Common Mistake	Example of mistake
1. Have 2 values to compare that are numbers (floats or integers)	Comparing values that are not numbers. This would be an error and your code would not run.	`x = 1.0003` `y = "1.0004"`
2. Define a small positive number as a threshold value	Defining 0 or a negative number as a threshold. Your code would still run and your check of float equality would always result in `False`.	`x = 1.0003` `y = 1.0004` `thres = 0.0`
3. Determine if the absolute difference is less than the threshold value.	Not including `abs()` to check absolute difference. Not including this would make it possible to get a negative number, resulting in your code running but outputting `True` when it isn't supposed to!	`x = 1.0003` `y = 1.0004` `thres = 0.0001` `(x-y) < thres`

Let's practice writing code with float equality templates!

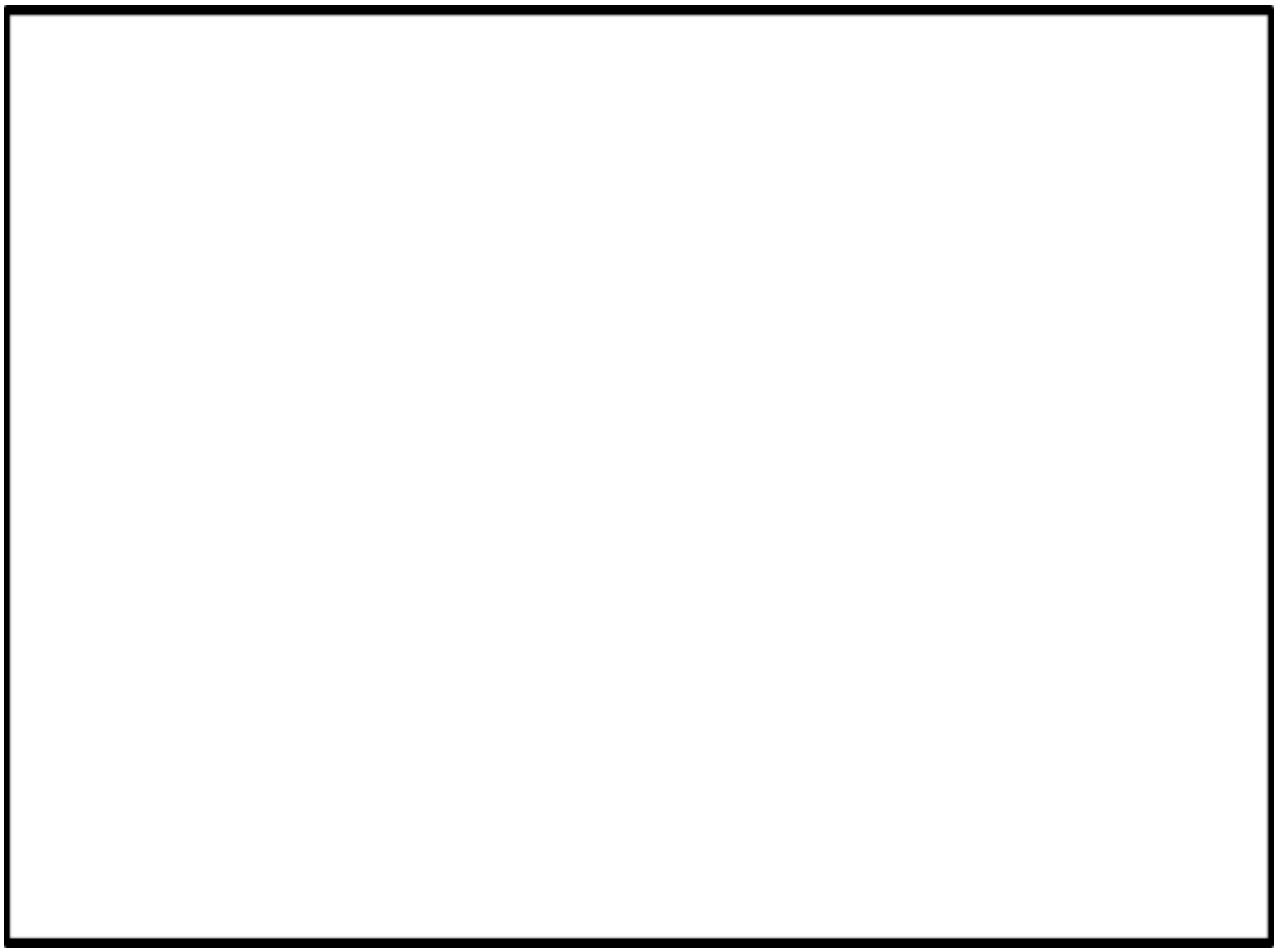
Initial here after reading page: _____

Practice: Write Float Equality Template

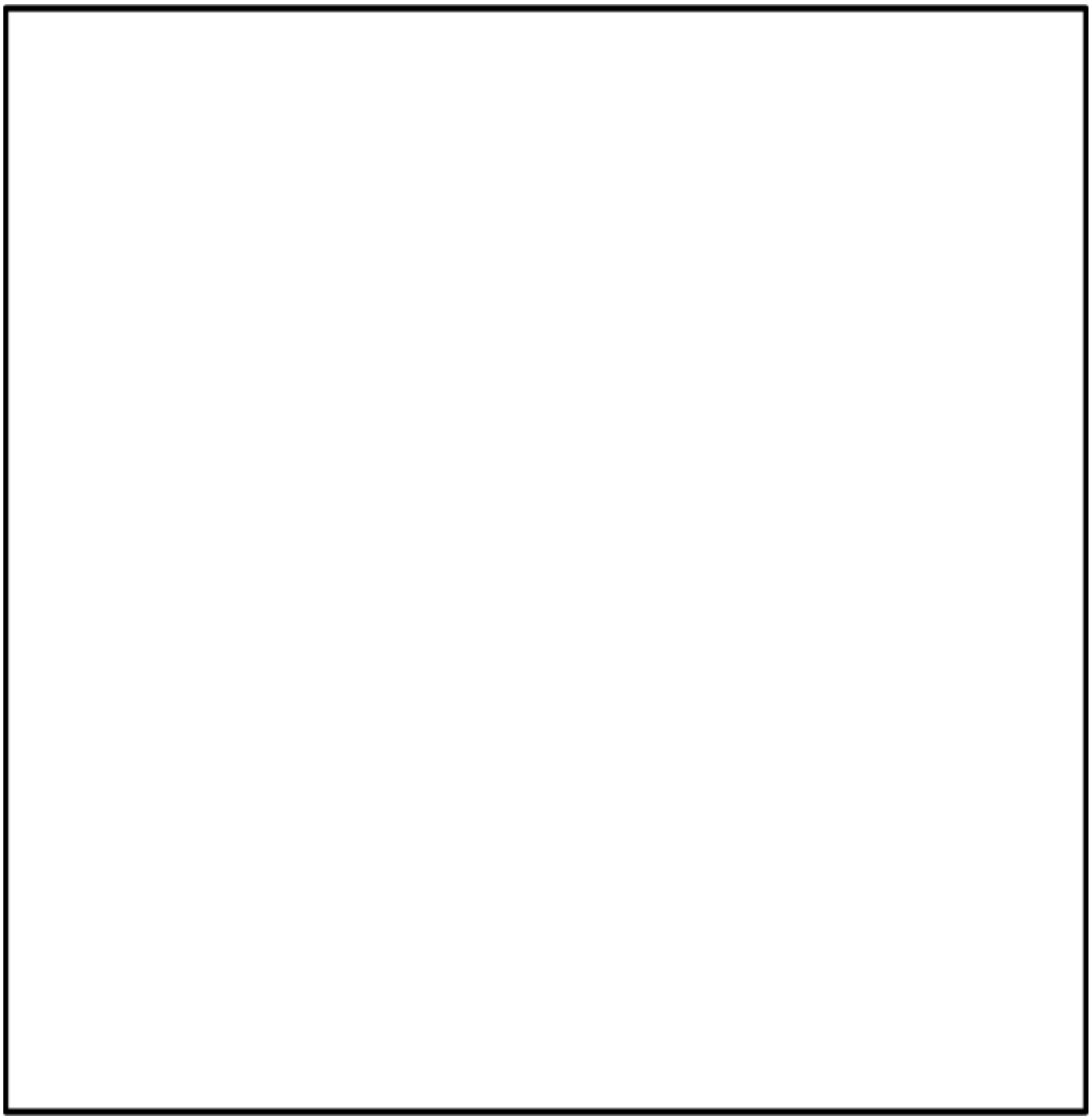
Assume variables `a`, `b`, and `c` are declared as integers or floats. Write code checks if `a`, `b`, and `c` are all approximately equal (within `0.001` of each other). If all variables are equal, then print `True`. Otherwise, print `False`.

Hint: Get the differences between `a` and `b`, the differences between `b` and `c`.

1) In plain English, describe a step-by-step plan for solving the problem:



2) Write the code to check if variables `a`, `b`, and `c` are all approximately equal (within `0.001` of each other).



3) After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Practice: Write Float Equality Template Solution

Check your answers to the following question:

Assume variables `a`, `b`, and `c` are declared as integers or floats. Write code checks if `a`, `b`, and `c` are all approximately equal (within `0.001` of each other). If all variables are equal, then print `True`. Otherwise, print `False`.

Hint: Get the differences between `a` and `b`, the differences between `b` and `c`.

Plan:

```
1) Store absolute differences between a and b, between b and c
2) Sum absolute differences
3) determine if sum of differences is less than threshold and print
```

Code:

```
thres = 0.001 # define variable for threshold

equal_ab = abs(a-b) # store absolute difference between a and b
equal_bc = abs(b-c) # store absolute difference between b and c

all_diff = equal_ab + equal_bc # store sum of differences

print(all_diff < thres) # determine if sum is less than threshold & print
```

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Reading Conditionals

Now that we know how to determine the relationship between values with relational operators, we want to do different things based on different relationships. In example, if it is cold outside then I will bring my raincoat. Or if I don't have any homework tonight, then I will meet up with my friends.

Conditional statements (also known as *if-statements*) enable different code to execute based on a given relationship.

Say you wanted to buy a soda but would only but it if it cost $1 or less. You could use a conditional statement to help you make this decision. Let's say the variable cost holds a float and was declared earlier. To determine whether we should buy the soda, we would write this code:

```
if(cost <= 1.00):
    print("buy the soda!")
```

So, we have the word "if" followed by a relational operation, known as a **condition** (`cost <= 1.00`), in parenthesis. If that condition evaluates to True, then all of the indented code under the if statement (just a print statement in this case) would execute. If the condition does not evaluate to True (evaluates to False), then the indented code would not have run.

If we wanted to run different code if the condition executes to false, we would add an else condition:

```
if(cost <= 1.00):
    print("buy the soda!")
else:
    print("do NOT buy the soda.")
    print("I repeat, do NOT buy it!")
```

If the condition evaluated to false (the cost of the soda was greater than 1.00), then all of the code under the else statement (2 print statements) would have run. This is because the computer executes all of code in the indented block under the else statement when the condition evaluates to false.

So to summarize, conditional statements run the indented code under the if statement if the condition is true. If the condition is not true, then the computer skips the indented code under the if statement and instead runs the indented code under the else statement.

Let's walk through this code:

Initial here after reading page: _____

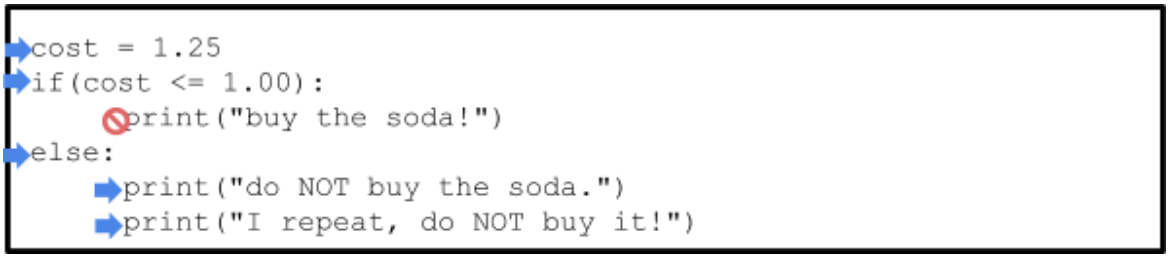
```
cost = 1.25
if(cost <= 1.00):
      print("buy the soda!")
else:
      print("do NOT buy the soda.")
      print("I repeat, do NOT buy it!")
```

Here, the cost variable is set to 1.25. We then check the if statement and find it to be false. So, we skip the indented code under the if statement and jump to the else statement. We then run the indented code under the else statement.

The output of running this code:

```
do NOT buy the soda
I repeat, do NOT buy it!
```

Here is the same code block with blue arrows to show the lines of code that execute and a red circle to show the line of code that did not execute.



Let's change the cost of the soda and look at that example again:

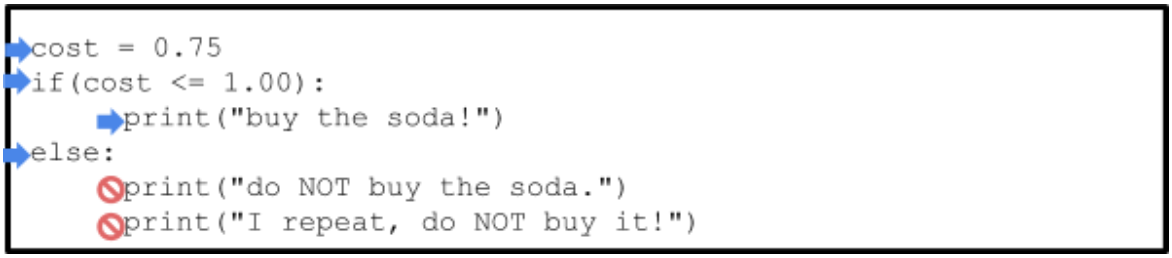
```
cost = 0.75
if(cost <= 1.00):
      print("buy the soda!")
else:
      print("do NOT buy the soda.")
      print("I repeat, do NOT buy it!")
```

Here, the cost is low enough so the condition in the if statement is true. So, the print statement that is indented under the if statement executes and the else statement is skipped.

The output of running this code:

```
buy the soda!
```

The annotated code block shows that the indented code under the else condition was skipped.



Initial here after reading page: _____

Else-if statements: when there are more than 2 options

Basic conditionals with an if and else condition help us make decisions between two options. But what if we wanted to introduce a 3rd option? For that, we'll need *else if statements*.

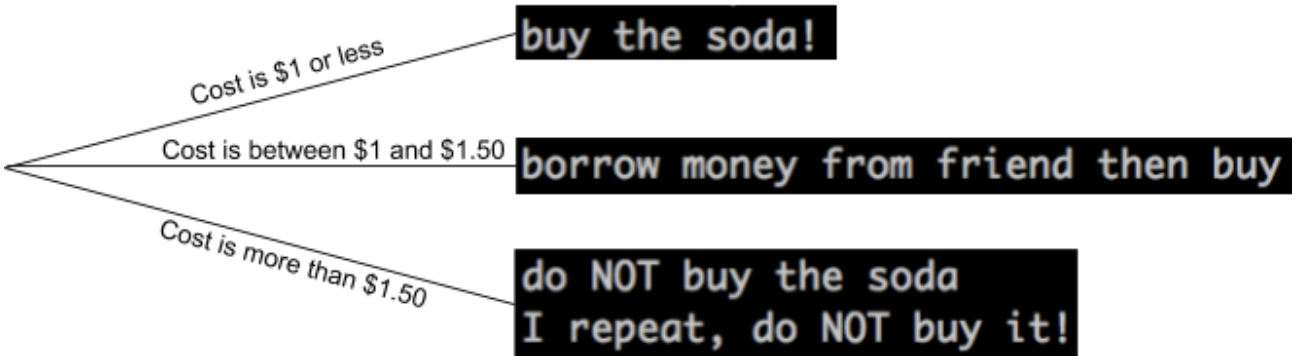
Say you have a friend who could lend you $0.50. So this way if the soda costs up to $1.50, you can still afford it. But if the soda costs more than $1 and less than or equal to $1.50, you'd need to ask you friend for the money. This introduces a third option. To account for this third option, we can introduce an **else-if statement**.

Else-if statements exist after the if statement and before the else statement. If the if condition evaluates to false, then the else-if condition is checked. If it is true, then the indented code under the else-if condition is evaluated and the else condition is skipped. If if the else-if condition also evaluates to false, then the else condition is run.

Let's see an example:

```
cost
if(cost <= 1.00):
      print("buy the soda!")
elif(cost <= 1.50):
      print("borrow money from friend then buy")
else:
      print("do NOT buy the soda.")
      print("I repeat, do NOT buy it!")
```

This code is equivalent to the diagram below, where the cost of the soda determines which indented block of code is executed. Notice how in each condition, only 1 condition is executed. So if a soda was $0.75, only `buy the soda!` would print out even though a cost of 0.75 would also make the else if condition true. This is because at most 1 condition will execute in any conditional statement.



Initial here after reading page: _____

Reading Conditionals Practice

Read through the following blocks of code. Cross out the lines of code that do not execute. Then, determine what the code would output.

```
friend = "juan"
temp = 30
if(temp <= 40 and friend == "sue"):
      print("Bring 2 jackets.")
elif(temp<=35):
      print("Bring 1 jacket.")
else:
      print("You don't need a jacket!")
```

Output:

```

```

```
num_people = 11
seats_per_table = 4
extra_chairs = 0
max_chairs = 2

if(num_people % seats_per_table > 0):
    extra_chairs = num_people % seats_per_table
else:
    print("No extra chairs needed.")

if(extra_chairs>0 and extra_chairs <= max_chairs):
    print("We'll need extra chairs")
else:
    print("We don't have enough chairs.")
```

Output:

```

```

Initial here after reading page: _____

Reading Conditionals Practice Solutions

Read through the following blocks of code. Cross out the lines of code that do not execute. Then, determine what the code would output.

```
friend = "juan"
temp_f = 30
if(temp <= 40 and friend == "sue"):
      print("Bring 2 jackets.")
elif(temp<=35):
      print("Bring 1 jacket.")
else:
      print("You don't need a jacket!")
```

Output:

```
Bring 1 jacket.
```

```
num_people = 11
seats_per_table = 4
extra_chairs = 0
max_chairs = 2

if(num_people % seats_per_table > 0):
    extra_chairs = num_people % seats_per_table
else:
    print("No extra chairs needed.")

if(extra_chairs>0 and extra_chairs <= max_chairs):
    print("We'll need extra chairs")
else:
    print("We don't have enough chairs.")
```

Output:

```
We don't have enough chairs.
```

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Writing Conditionals

When writing conditionals, we should be careful of both writing the code correctly but also making sure the logic in the conditionals are correct.

To write an if statement correctly, there are a few things to keep in mind:

Rule	Broken Code	Fixed Code
The condition in an if statement has a colon at the line.	`if x < 2` `    print ("yes")`	`if (x < 2):` `    print ("yes")`
Indent the code that should run if a condition is true	`if(x<2):` `print ("yes")`	`if(x<2):` `    print ("yes")`
Other conditions (e.g. `elif`, `else`) for a given if statement should be at the same level of indentation.	`if(x<2):` `    print ("yes")` `    elif(x<5):` `        print("ok")` `    else:` `        print("too big")`	`if(x<2):` `    print ("yes")` `elif(x<5):` `    print("ok")` `else:` `    print("too big")`
Else if statements begin with `elif`	`if(x<2):` `    print ("yes")` `else if(x<5):` `    print("ok")`	`if(x<2):` `    print ("yes")` `elif(x<5):` `    print("ok")`

Initial here after reading page: _____

We have to be careful when writing conditional statements because making minor changes can influence how our conditionals work. Here are some things to remember:

Rule	Dangerous Code	Explanation
Every if statement will run.	`cost=0.45` `if(cost < 0.5):` `  print("buy it!")` **`if(cost <0.75):`** `  print("borrow money to buy")` `else:` `  print("don't buy it")`	In this example, 2 print statements will output because the conditional statement for the 2 if statements are true. Replacing the bold if statement with an else-if would fix this logical error.
If statements are required. Else-if and else statements are not required.	`name = "sue"` **`elif`**`(name == "bob"):` `  print("Bob is here!")` `else:` `  print("Bob is not here.")`	This code would not run because there is no if statement to start the conditional. Replacing the `elif` with an `if` would make this code run.
We can use the `and` operator with conditionals.	`will_rain = True` `going_outside = False` `if(will_rain and going_outside):` `  print("bring raincoat")` `else:` `  print("no worries")`	In this example, "`no worries`" is printed because `going_outside` is false, so the else statement is executed.

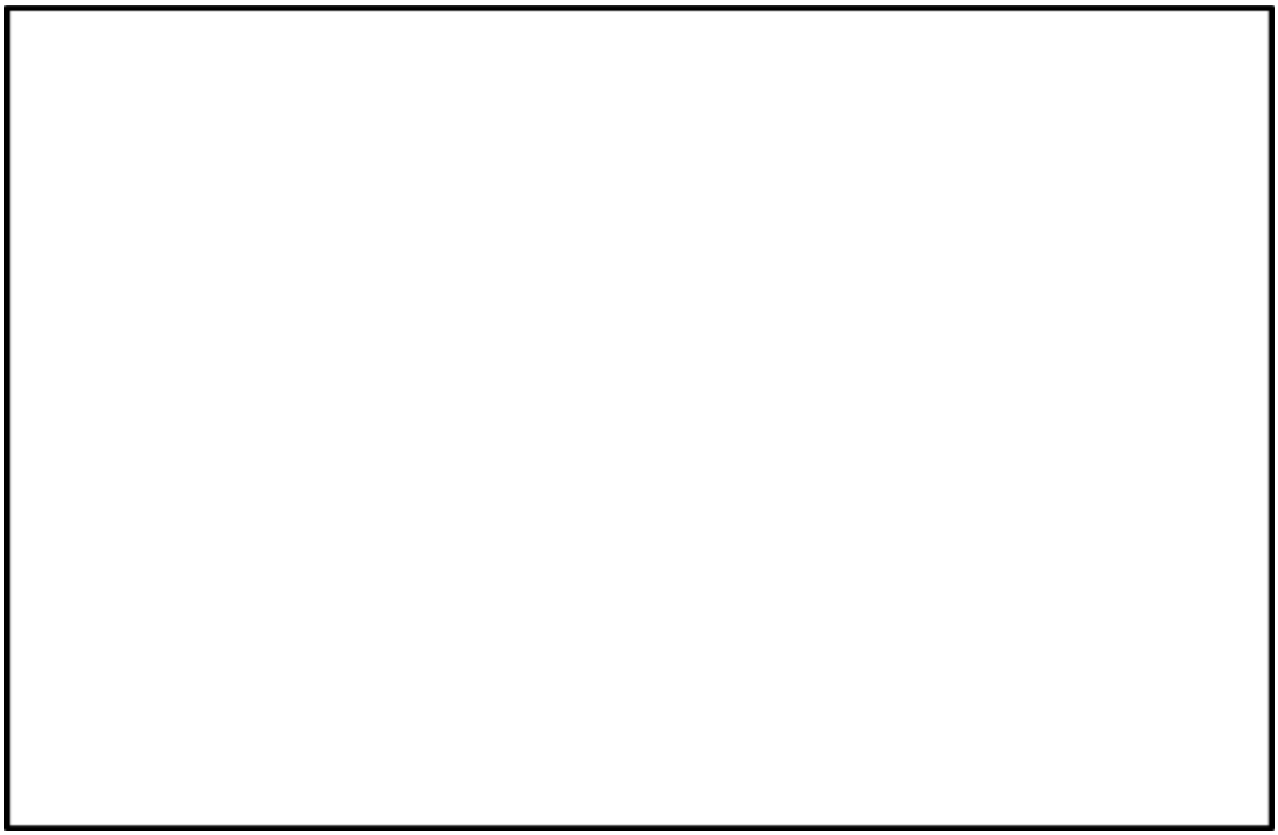
Let's get some practice writing conditionals!

Initial here after reading page: _____

Writing Conditionals Practice

Write code that does the following:

- *Declare a variable* `profit` *and set it to* `87`*.*
- *Declare a variable* `cost` *and set it to* `75`*.*
- *Uses an if statement to check if* `profit` *is greater than* `cost`*.*
  If this condition evaluates to true, then the following code evaluates:
  - *Declare a variable* `money_made` *and set it to the* `profit` *minus the* `cost`*.*
  - *Print "*`profit`*"*
  - *Print the value stored in* `money_made`
- *Uses an else if condition where the condition checks if* `profit` *and* `cost` *are equal.*
  If this condition evaluates to true, then the following code evaluates:
  - *Print "*`break even`*"*
- *Uses an else condition that would evaluate the following code:*
  - *Print "*`lost money`*"*



After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Writing Conditionals Practice Solutions

Check your code for the following problem:

Write code that does the following:

- *Declare a variable* `profit` *and set it to* `87`*.*
- *Declare a variable* `cost` *and set it to* `75`*.*
- *Use an if statement to check if* `profit` *is greater than* `cost`*.*
  If this condition evaluates to true, then the following code evaluates:
  - *Declare a variable* `money_made` *and set it to the* `profit` *minus the* `cost`*.*
  - *Print "*`profit`*"*
  - *Print the value stored in* `money_made`
- *Use an else if condition where the condition checks if* `profit` *and* `cost` *are equal.*
  If this condition evaluates to true, then the following code evaluates:
  - *Print "*`break even`*"*
- *Use an else condition that would evaluate the following code:*
  - *Print "*`lost money`*"*

```
profit = 87  # define var and store int
cost = 75    # define var and store int

if (profit > cost):                  # check if profit is bigger
      money_made = profit - cost     # store amount of money made
      print("profit")                # print string literal
      print(money_made)              # print value in money_made
elif(profit == cost):                # else if profit & cost equal
      print("break even")            # print string literal
else:                                # else condition
      print("lost money")            # print string literal
```

Check the boxes that apply:

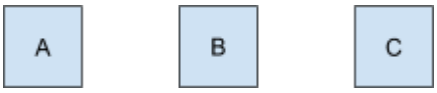
- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

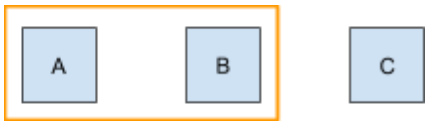
Read Conditionals Template: Finding max or min

Say we had numbers and we wanted to print the smallest number. We can use a simple if/else to do this for 2 numbers. But what if we had more than 2 numbers? To do this, we can use what we know about the `and` operator and if statements and else if statements. Let's see a conceptual example:

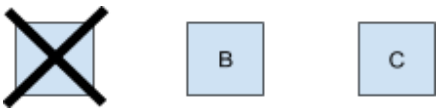
Say I had 3 variables which stored numbers and I wanted to find the largest value but I could only look at at most 2 variables at a time. How could I do this?



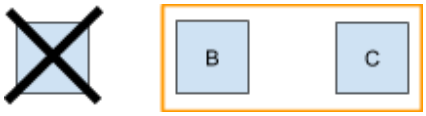
Well I could start by comparing A and B.



If I see A is greater than B, then I should also see if A is greater than C. If A is greater than B and C, then I'm done. If A is not greater than B and C, then I know the largest value is not A. I can then ignore A moving forward.



My next step is to compare B and C.



If B is larger than C, then I know B is the largest number. I do not have to compare A and B again because I already checked A previously! If we find that B is not larger than C, we can say that B is not the largest number and remove it.

If we found that the largest number was not A or B, then it must be C! Notice we started with 3 numbers and had to check 1 number against the other 2. If it wasn't what we were looking for then we removed it. We then checked 1 of the remaining numbers against the other 1. If it wasn't the one we were looking for, then we removed it. We then only had 1 number left and 0 numbers to compare against. If that was the case, then that last number must be the one we're looking for.

This process of checking a value then removing it from future checks and repeating this process with the remaining numbers is how we find the maximum or minimum number from multiple options. To see this in code:

Initial here after reading page: _____

```
if (a > b and a > c):
    print(a)
elif(b > c):
    print(b)
else:
    print(c)
```

Similar to the example on the previous page, we used the `and` operator to see if `a` is the largest number. If it is not, then we can ignore it and we move on to the else if condition to see if `b` is the largest. If that is not true, then we know in the else condition that `c` is the largest.

If we changed all the > signs to <, we can use the same template to find the minimum value!

So, to find the maximum (largest) or minimum (smallest) value, we do the following:

1. Use if statements to check 1 value against all other remaining values
    a. We may need a compound conditional statement (using `and`)
2. Ignore the value we just checked and repeat step 1 if there are at least 2 remaining values
3. If there are no more values to compare against, then we reach our else condition.

Let's practice reading the maximum or minimum template!

Practice reading find max or min template

For each chunk of code, read the code and fill in the variable table to identify values for variables a, b, and c which would result in the code incorrectly finding the minimum value. Then, identify the error from the multiple choice options.

1)

```
if (a < b and a < c):
    print(a)
elif(c < b):
    print(b)
else:
    print(c)
```

Variable Name	Variable Value
a	
b	
c	
Output:	

Select one:

(a) A value is not eliminated correctly.
(b) The value being printed does not match the value being checked in the conditional.
(c) All values are not eliminated.

2)

```
if (a < b):
    print(a)
elif (a < c):
    print(a)
elif(b < c):
    print(b)
else:
    print(c)
```

Variable Name	Variable Value
a	
b	
c	
Output:	

Select one:

(a) A value is not eliminated correctly.
(b) The value being printed does not match the value being checked in the conditional.
(c) All values are not eliminated.

3)

```
if (a < b and a < c):
    print(a)
elif(b < c):
    print(b)
```

Variable Name	Variable Value
a	
b	
c	
Output:	

Select one:

(a) A value is not eliminated correctly.
(b) The value being printed does not match the value being checked in the conditional.
(c) All values are not eliminated.

Initial here after reading page: _____

Practice reading find max or min template solutions

For each chunk of code, determine if it prints the minimum value for variables `a`, `b`, and `c` which each store a number. If does is not, identify the error from the multiple choice options.

1)

```
if (a < b and a < c):
    print(a)
elif(c < b):
    print(b)
else:
    print(c)
```

Variable Name	Variable Value
a	2
b	3
c	1
Output:	3

Select one:
(b) The value being printed does not match the value being checked in the conditional.

2)

```
if (a < b):
    print(a)
elif (a < c):
    print(a)
elif(b < c):
    print(b)
else:
    print(c)
```

Variable Name	Variable Value
a	2
b	3
c	1
Output:	2

Select one:
(a) <u>A value is not eliminated correctly.</u>

3)

```
if (a < b and a < c):
    print(a)
elif(b < c):
    print(b)
```

Variable Name	Variable Value
a	2
b	3
c	1
Output:	

Select one:
(c) <u>All values are not eliminated.</u>

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Writing Find Max or Min Templates

When writing code to find the maximum or minimum value, it is important to keep a few things in mind:

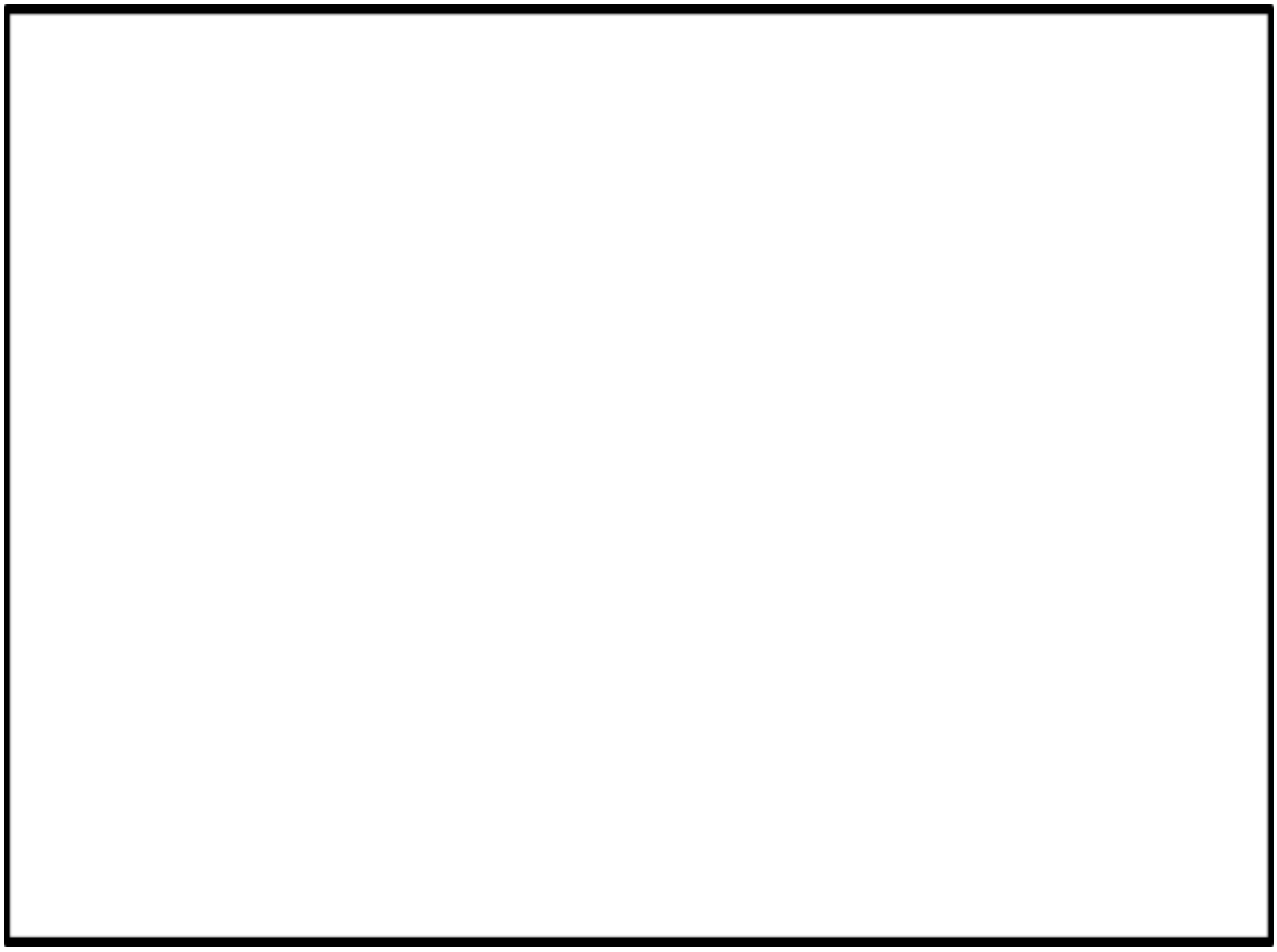
Rule	Bad Code	Explanation
Compare a value against all values that have NOT been eliminated.	`if (a > b and a > c):` `  print(a)` `elif(b > c **and b > a):**` `  print(b)` `else:` `  print(c)`	The variable `a` has already been eliminated and does not need to be compared against again. This leads to unnecessary confusion.
If there are more than 2 values to compare, you will need combine if statements with `and`.	`if (a > b):` `**if (a > c):**` `  print(a)` `elif(b > c):` `  print(b)` `else:` `  print(c)`	The bold code should be included as a compound conditional in the if statement above it: `if(a>b and a>c)`
Make sure the conditional statement matches the variable you are outputting.	`if (a > b and a > c):` `  print(**c**)` `elif(b > c):` `  print(b)` `else:` `  print(c)`	The first if statement is checking to see if the variable `a` is the maximum but this code prints the variable `c`.

Initial here after reading page: _____

Practice Writing Find Max or Min Templates

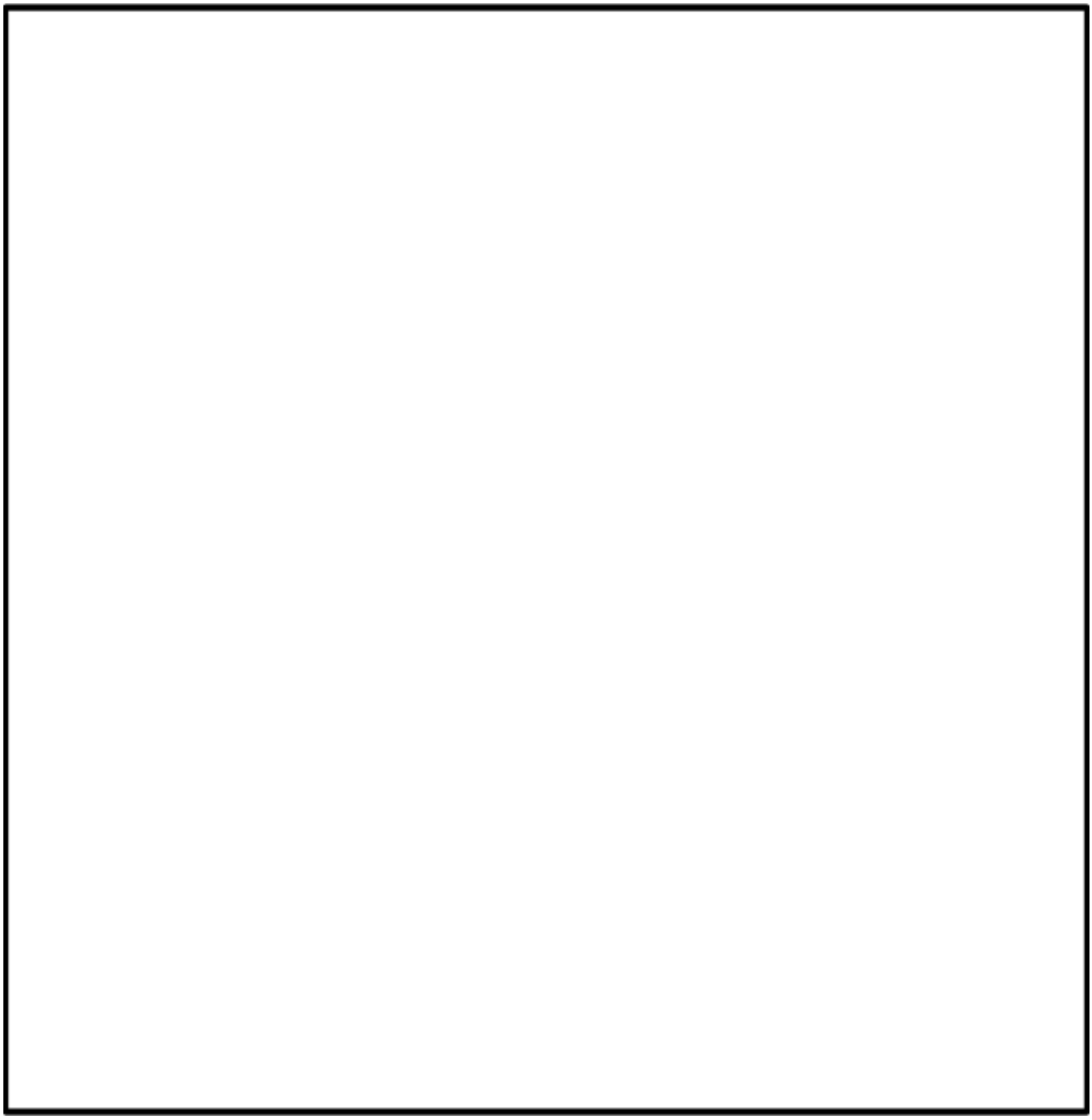
Given the variables `x`, `y`, and `z` which all store numbers, print the sum of the maximum and minimum values.

1) In plain English, describe a step-by-step plan for solving the problem:



Initial here after reading page: _____

2) Write the code that prints the sum of the maximum and minimum values of variables `x`, `y`, and `z`, which all store numbers.



3) After you're done writing code, write a comment next to each line describing what the line does <u>in your own words</u>.

Initial here after reading page: _____

Practice Writing Find Max or Min Templates Solutions

Check your solutions for the following problem:

Given the variables `x`, `y`, and `z` which all store numbers, print the sum of the maximum and minimum values.

Plan:

1) Use max/min template (if/elif/else statements) to find min value and store it
2) Use max/min template (if/elif/else statements) to find min value and store it
3) Print sum of max and min values

Code:

```
if(x<y and x<z):          # check if x is smallest
    min_val = x           # if so, set min to x
elif(y<z):                # check if y is smallest
    min_val = y           # if so, set min to y
else:                     # else condition
    min_val = z           # set min to z

if(x>y and x>z):          # check if x is max
    max_val = x           # if so, set x to max
elif(y>z):                # check if y is max
    max_val = y           # if so, set y to max
else:                     # else condition
    max_val = z           # set min to z

print(min_val + max_val) # print sum of max and min vals
```

Check the boxes that apply:

- ATTEMPT: I was able to provide answer(s) without looking at the solutions. Yes: ☐ / No: ☐
- AGREEMENT: All of my answers are in agreement with the solution Yes: ☐ / No: ☐

Initial here after reading page: _____

Summary

That is the conclusion of this instruction. To sum up, you have learned how about the following Python concepts:

Concept	Description	Examples
Data Types	Different classifications of data (string, boolean, integer, float)	`"Hello"` `True` `3` `3.1`
Variables	Store values to be used later. Can be updated.	`cost = 1.50` `cost = 1`
Arithmetic operations	Math operations to be done between numbers	`8 / 2` `(3 + 1) * 4` `7 % 3`
Print statements	Output values to be displayed on the console	`print("hello world!")`
Relational operations	Determine if a relationship between two values is valid or not	`3 < 7` `"hi" == "HI"`
Conditionals	Execute different code based on condition	`cost = 1.4` `if (cost < 1):` `  print("buy it!")` `else:` `  print("do not buy")`

Initial here after reading page: _____