Figure out what's the relationship between the strings "cat" and "Cat" by replacing the plus sign with comparison operators.

print("cat" > "Cat")

RunReset



"cat" equals "Cat"



"cat" is smaller than "Cat"



"cat" is larger than "Cat"

**Correct**

You nailed it! In Python uppercase letters are alphabetically sorted before lowercase letters.

Comparison Operators

In Python, we can use comparison operators to compare values. When a comparison is made, Python returns a boolean result, or simply a True or False.

* To check if two values are the same, we can use the equality operator: **==**
* To check if two values are not the same, we can use the not equals operator: **!=**

We can also check if values are greater than or lesser than each other using **>** and **<**. If you try to compare data types that aren’t compatible, like checking if a string is greater than an integer, Python will throw a **TypeError**.

We can make very complex comparisons by joining statements together using logical operators with our comparison operators. These logical operators are **and**, **or**, and **not**. When using the **and** operator, both sides of the statement being evaluated must be true for the whole statement to be true. When using the **or** operator, if either side of the comparison is true, then the whole statement is true. Lastly, the **not** operator simply inverts the value of the statement immediately following it. So if a statement evaluates to True, and we put the **not** operator in front of it, it would become False.

**The is\_positive function should return True if the number received is positive, otherwise it returns None. Can you fill in the gaps to make that happen?**



def is\_positive(number):

if number > 0:

return True

RunReset

is\_positive(-5) returned None

is\_positive(0) returned None

is\_positive(13) returned True

Congrats! You've now written

your first conditional block. Great work!

## if Statements Recap

We can use the concept of **branching** to have our code alter its execution sequence depending on the values of variables. We can use an if statement to evaluate a comparison. We start with the if keyword, followed by our comparison. We end the line with a colon. The body of the if statement is then indented to the right. If the comparison is **True**, the code inside the if body is executed. If the comparison evaluates to **False**, then the code block is skipped and will not be run.

**The is\_positive function should return True if the number received is positive and False if it isn't. Can you fill in the gaps to make that happen?**

def is\_positive(number):

if number > 0:

return True

else:

return False

RunReset

is\_positive(-5) returned False

is\_positive(0) returned False

is\_positive(13) returned True

Well done, you! You're starting to

master conditional expressions!

## else Statements and the Modulo Operator

We just covered the if statement, which executes code if an evaluation is true and skips the code if it’s false. But what if we wanted the code to do something different if the evaluation is false? We can do this using the else statement. The else statement follows an if block, and is composed of the keyword else followed by a colon. The body of the else statement is indented to the right, and will be expected if the above if statement doesn’t execute.

We also touched on the modulo operator, which is represented by the percent sign: **%**. This operator performs integer division, but only returns the remainder of this division operation. If we’re dividing 5 by 2, the quotient is 2, and the remainder is 1. Two 2s can go into 5, leaving 1 left over. So 5%2 would return 1. Dividing 10 by 5 would give us a quotient of 2 with no remainder, since 5 can go into 10 twice with nothing left over. In this case, 10%2 would return 0, as there is no remainder.

**The number\_group function should return "Positive" if the number received is positive, "Negative" if it's negative, and "Zero" if it's 0. Can you fill in the gaps to make that happen?**

def number\_group(number):

if number>0:

return "Positive"

elif number<0:

return "Negative"

else:

return "zero"

print(number\_group(10)) #Should be Positive

print(number\_group(0)) #Should be Zero

print(number\_group(-5)) #Should be Negative

RunReset

Here is your output:

Positive

zero

Negative

Well done! You're progressing so fast!

## More Complex Branching with elif Statements

Building off of the if and else blocks, which allow us to branch our code depending on the evaluation of one statement, the elif statement allows us even more comparisons to perform more complex branching. Very similar to the if statements, an elif statement starts with the elif keyword, followed by a comparison to be evaluated. This is followed by a colon, and then the code block on the next line, indented to the right. An elif statement must follow an if statement, and will only be evaluated if the if statement was evaluated as false. You can include multiple elif statements to build complex branching in your code to do all kinds of powerful things!

## Conditionals Cheat Sheet

## Conditionals Cheat Sheet

In earlier videos, we took a look at some of the built-in Python operators that allow us to compare values, and some logical operators we can use to combine values. We also learned how to use operators in if-else-elif blocks.

It’s a lot to learn but, with practice, it gets easier to remember it all. In the meantime, this handy cheat sheet gives you all the information you need at a glance.

### **Comparison operators**

* a == b: a is equal to b
* a != b: a is different than b
* a < b: a is smaller than b
* a <= b: a is smaller or equal to b
* a > b: a is bigger than b
* a >= b: a is bigger or equal to b

### **Logical operators**

* a and b: True if both a and b are True. False otherwise.
* a or b: True if either a or b or both are True. False if both are False.
* not a: True if a is False, False if a is True.

**Branching blocks**

In Python, we branch our code using if, else and elif. This is the branching syntax:

if condition1:

if-block

elif condition2:

elif-block

else:

else-block

Remember: The if-block will be executed if condition1 is True. The elif-block will be executed if condition1 is False and condition2 is True. The else block will be executed when all the specified conditions are false.

## Practice Quiz: Conditionals

**TOTAL POINTS 5**

1.Question 1

What's the value of this Python expression: (2\*\*2) == 4?



4



2\*\*2



True



False

**Correct**

You nailed it! The conditional operator == checks if two values are equal. The result of that operation is a boolean: either True or False.

**1 / 1 point**

2.Question 2

Complete the script by filling in the missing parts. The function receives a name, then returns a greeting based on whether or not that name is "Taylor".



def greeting(name):

if name == "Taylor":

return "Welcome back Taylor!"

else:

return "Hello there, " + name

print(greeting("Taylor"))

print(greeting("John"))

RunReset

**Correct**

Great work! You're getting the hang of conditionals in Python.

**1 / 1 point**

3.Question 3

What’s the output of this code if number equals 10?



if number > 11:

print(0)

elif number != 10:

print(1)

elif number >= 20 or number < 12:

print(2)

else:

print(3)

2

**Correct**

Right on! Our number is 10, which is smaller than 12, so it matches that condition.

**1 / 1 point**

4.Question 4

Is "A dog" smaller or larger than "A mouse"? Is 9999+8888 smaller or larger than 100\*100? Replace the plus sign in the following code to let Python check it for you and then answer.

print("A dog" < "A mouse")

print(9999+8888 > 100\*100)

RunReset



"A dog" is larger than "A mouse" and 9999+8888 is larger than 100\*100



"A dog" is smaller than "A mouse" and 9999+8888 is larger than 100\*100



"A dog" is larger than "A mouse" and 9999+8888 is smaller than 100\*100



"A dog" is smaller than "A mouse" and 9999+8888 is smaller than 100\*100

**Correct**

You got it! Keep getting Python to do the work for you.

**1 / 1 point**

5.Question 5

If a filesystem has a block size of 4096 bytes, this means that a file comprised of only one byte will still use 4096 bytes of storage. A file made up of 4097 bytes will use 4096\*2=8192 bytes of storage. Knowing this, can you fill in the gaps in the calculate\_storage function below, which calculates the total number of bytes needed to store a file of a given size?

def calculate\_storage(filesize):

block\_size = 4096

# Use floor division to calculate how many blocks are fully occupied

full\_blocks = int(filesize/4096)

# Use the modulo operator to check whether there's any remainder

partial\_block\_remainder = filesize % 4096

# Depending on whether there's a remainder or not, return

# the total number of bytes required to allocate enough blocks

# to store your data.

if partial\_block\_remainder > 0:

return (full\_blocks+1)\*4096

return (full\_blocks\*4096)

print(calculate\_storage(1)) # Should be 4096

print(calculate\_storage(4096)) # Should be 4096

print(calculate\_storage(4097)) # Should be 8192

print(calculate\_storage(6000)) # Should be 8192

RunReset

**Correct**

Awesome! Those were some complicated calculations that you

needed to do, but you did it!