Conditional Statements & Loops: If-else Control Structures in Python

Learners will explore implementations of the order of precedence of operators, using if-elif-else statements to evaluate multiple conditions and conversions between various data types in Python, in this 15-video course. Key concepts covered here include how conditions in Python work, and how to evaluate conditions by involving primitive data types using if statements and complex data types using if statements. Next, evaluate multiple conditions for decision making with nested control structures; identify how to use the if-else statement to make decisions involving complex data types such as lists, tuples, and dictionaries; and learn how to convert an integer to a float and a float or an integer to a string, and vice-versa. Learners then observe how to convert primitive data types to complex data types, to convert between various complex data types, and to convert between various complex data types and view base conversions with Python built-in functions; and to solve various programming problems with Python built-in methods. Finally, you will learn to solve various programming problems by using if-elif-else statements and nested if-else statements.

Table of Contents

Course Overview

[Video description begins] *Topic title: Course Overview. The host for this session is Yukti Kalra. She is a Product Manager.* [Video description ends]

Hi, and welcome to this course if-else conditional control structures in Python. My name is Yukti Kalra and I will be your instructor for this course. A little about myself first, I have a background in business and an MBA in analytics. I also possess extensive experience in building machine learning and statistical models, as well as in big data and cloud computing. I currently work for Loonycorn, a studio for high-quality video content. For all its popularity, Python is at heart, a programming language, just like any other. A Python program, just like a program in any other language, can be expressed as a combination of different control structures of three main types. The default sequential control structure in which statements execute one after another. Conditional control structures in which conditions are evaluated before deciding what the program does next. And iterative or repetitive control structures in which statements are repeatedly executed until a terminating condition evaluates to true. In this course, our focus is on the second of these control structures, namely the conditional control structure. Most programming languages have several different types of conditional control structures such as if, else, do, while, and Switch Case statements. And in this course we focus on if, else conditions which are among the most common and popular control structures. If, else statements are ubiquitous. They are used in domains ranging from hardware assembly language programming to financial formulae in Excel and other spreadsheet programs. By the end of this course, you will have a solid practical and theoretical understanding of the if, else, elif construct in Python. And we'll be able to use it to direct your programs control flow into the appropriate code block based on appropriate preconditions that need to be satisfied. This will take you a long way towards building complex programs by combining functions and other code blocks in meaningful ways.

Python Conditions

[Video description begins] *Topic title: Python Conditions. The presenter is Yukti Kalra. [Video description ends]*

Look at the problem on the screen.

[Video description begins] The PythonConditions page opens in the jupyter notebook. It is divided into three parts. The first part is a menu bar. It includes File, Edit, Insert, and View options. The second part is a toolbar. The third part is a content pane. It contains several cells. The first cell contains the code, (5 > 3) and (4 < 6). The output reads: True. The second cell contains the code, 5 > 3 and 4 < 2. The output reads: False. The third cell contains the code, 10 == 10 == True. [Video description ends]

What will this return?

[Video description begins] *She executes the code, 10 == 10 == True. The output reads: False. She highlights the content and output of all the cells.* [Video description ends]

We may think we know all about comparisons, but there are some subtle points, word exploring. So far we've learned basic constructs of programming, variables to store value, and various data types. We're now ready to step into the real programming. We'll start off by looking at how we can check conditions within our program. These conditions can be used in several ways, most commonly in if-else statements and loops. Programming is all about making decisions based on the conditions that are evaluating. What are conditions? Conditions are statements that return true or false. Is 10 == 10?

[Video description begins] *In a cell, she enters the code, 10 == 10.* [Video description ends]

We all know the answer to this. This will return true. Notice the use of the double equal to symbol, this is extremely important. In Python, a single equal to operator is used for assignment, and the double equal to symbol is used for equality. We are familiar with a single equal to operator. There, we perform a computation and store it in a variable. That's where we use the single equal to operator. Then we want to perform the equality operation as you see on the screen, we use the double equal to symbol. What we want to do here is we want to check if the value on the left is equal to the value on the right. Yes, 10 == 10, so this condition evaluates to true. Let's evaluate another condition is 10 == 20?

[Video description begins] *In a cell, she enters the code, 10 == 20.* [Video description ends]

And as you would expect, this evaluates to false. Is the integer 20 equal to the string "Hello"? Well, that's false too.

[Video description begins] *In a cell, she enters the code, 20 == "Hello". [Video description ends]*

If you were to use the single equal to operator instead of the double equal to symbol for equality evaluation, you would get an error.

[Video description begins] In a cell, she enters the code, 20 = "Hello". [Video description ends]

This single equal to symbol is wrong here because you cannot assign to the left-hand side. You cannot assign the string "Hello" to the value 20 because it's not a variable.

[Video description begins] *She executes the code, 20 = "Hello". An error message SyntaxError: can't assign to literal is displayed.* [Video description ends]

The error you get is fairly clear about it.

[Video description begins] *She highlights the SyntaxError: can't assign to literal.* [Video description ends]

It's a syntax error, you're using the wrong syntax.

[Video description begins] *In a cell, she enters the code, "Hello" == 20.* [Video description ends]

You can swap the left-hand side and the right-hand side when you're performing equality.

[Video description begins] *She executes the code, "Hello" == 20. The output reads: False.* [Video description ends]

This is the same equality check that we had performed earlier.

[Video description begins] *In a cell, she enters the code, "Hello" == "Hello".* [Video description ends]

You can perform equality checks with strings as well. Is this string "Hello" equal to the string "Hello"? Well, that's true. But remember, strings in Python are case sensitive. Using the double equal to symbol is extremely important when you're performing equality checks with variables. Here, a == b basically checks whether the value stored in the variable a is equal to the value stored in the variable b. This is a conditional equality check. A teeny, little change here, the use of single equal to completely changes the nature of the statement. This is then assignment operation a = b will make the Python interpreter take the value of b and store it in the variable a. This is completely different from the equality check that we intended to do. So make sure that you're very careful.

[Video description begins] *In a cell, she enters the code, 20 != 20.* [Video description ends]

Now that you've understood equality checks, Python also has inequality checks. If the value of two operands are not equal, then the condition becomes true. Notice the inequality operator in Python, it is an exclamation mark followed by an equal to symbol. The statement 20 != 20 evaluates to false. Is 20 != 10? This evaluates to true.

[Video description begins] *In a cell, she enters the code, 20 != 10. [Video description ends]*

The value 20 is not equal true the string "Hello", evaluates to true.

[Video description begins] In a cell, she enters the code, 20 != "Hello". [Video description ends]

The string "Hello" not equal to the string "Hello" evaluates to false because they are equal.

[Video description begins] *In a cell, she enters the code, "Hello" != "Hello".* [Video description ends]

Let's see how we can use variables in condition check.

[Video description begins] *In a cell, she enters the code, x = "Hello".* [Video description ends]

We assign the string "Hello" to the variable x.

[Video description begins] *In a cell, she enters the code, x == "Hello".* [Video description ends]

Now when you do an equality check of the string "Hello" with the variable x, it will evaluate to true because that is the value x holds. What will happen if we do an equality check of x with the string "Python"?

[Video description begins] *In a cell, she enters the code, x == "Python".* [Video description ends]

This evaluates to false as the variable x holds the string "Hello", not "Python".

[Video description begins] *She highlights the code, x = "Hello"*. [Video description ends]

Let's perform an inequality check with the variable x and the string "Python".

[Video description begins] *In a cell, she enters the code, x != "Python"*. [Video description ends]

This evaluates to true. Let's take a look at another comparison operator in Python, the less than operator.

[Video description begins] *In a cell, she enters the code, 10 < 20*. [Video description ends]

It evaluates to true when the value on the left-hand side is less than that on the right. Is 10 < 20?

[Video description begins] *In a cell, she enters the code, 10 < 9.[Video description ends]*

We all know the answer to that.

[Video description begins] In a cell, she enters the code, 10 <10.[Video description ends]

So it evaluates to true. Is 10 < 9? That is false. Is 10 < 10? That's false too.

[Video description begins] In a cell, she enters the code, 15 > 20. [Video description ends]

Let's take a look at the greater than operator. The greater than operator checks if the value on the left-hand side is greater than that on the right. Is 15 > 20? This evaluates to false.

[Video description begins] *In a cell, she enters the code, 20 <= 15*. [Video description ends]

In addition to the greater than operator and the less than operator, Python also supports less than equal to and greater than equal to operators. 20 <= 15 evaluates to false.

[Video description begins] *In a cell, she enters the code, 20 <= 20*. [Video description ends]

Now what will be the result of 20 <= 20. This evaluates to true.

[Video description begins] *In a cell, she enters the code, 20 < 20*. [Video description ends]

Let's check the condition 20 < 20 without the equal to operator, this evaluates to false. The condition above is only true because of the presence of the equal to operator in addition to the less then operator.

[Video description begins] *In a cell, she enters the code, 10 >= 20*. [Video description ends]

10 <= 20 evaluates to false. In Python, we can use expressions in conditions.

[Video description begins] *In a cell, she enters the code, 2 + 3 < 6*. [Video description ends]

Here we want to evaluate whether 2 + 3 < 6 or not. But how does the Python interpreter know that we want to perform the addition operations first and then the less than operator after that? There's a rule of precedence in Python. It guides the way in which the operations are carried out. The addition operator has a higher precedence than the less than operator, which is why 2 + 3 is evaluated first and then the less than operator. You can think of this rule to be something similar to the PEMDAS rule that you must have started in your high school math.

[Video description begins] *She highlights the link: https://docs.python.org/3/reference/expressions.html#operator-precedence*. [Video description ends]

You can go to the link below and find the table that summarizes the operator precedence in Python. This expression evaluates to true. Now we have 2 +3 within parentheses.

[Video description begins] *In a cell, she enters the code, (2 + 3) < 6.* [Video description ends]

Since the addition operator has higher precedence than less than operator, this statement is similar to the statement above, where the operands of addition bind force, hence this expression evaluates to true.

[Video description begins] *In a cell, she enters the code, 8 + 5 > 10 + 2.* [Video description ends]

Is 8 + 5 >10 + 2. Yes indeed, 13 is more than 12. Due to the higher precedence of the addition operation, the addition is calculated first on the either sides of the greater than operator, and then they're compared. Is 8 + 5 == 10 + 3?

[Video description begins] *In a cell, she enters the code, 8 + 5 == 10 + 3.* [Video description ends]

Yes, indeed it is, it evaluates to true. All the comparison operators have the same precedence. This automatically makes the precedence of the addition operator, higher than all the comparison operators.

[Video description begins] *In a cell, she enters the code, 2 * 5 > 10.* [Video description ends]

The precedence order of the multiplication operator is higher than that of the addition operator, the subtraction operator, and all the comparison operators. Therefore, this expression will evaluate to false.

[Video description begins] *In a cell, she enters the code, 2 * 5 >= 10.* [Video description ends]

2 x 5 >= 10 evaluates to true.

[Video description begins] *In a cell, she enters the code, 40 % 3 == 0.* [Video description ends]

Let's check if 40 modular 3 gives out the remainder 0, where the 40 is perfectly divisible by 3. And as you would've expected, this condition evaluates to false.

[Video description begins] *In a cell, she enters the code, 40 % 2 == 0.* [Video description ends]

Let's check if 40 is perfectly divisible by 2 or not. This evaluates to true. Let's swap the left-hand side and the right-hand side operands.

[Video description begins] *In a cell, she enters the code, 0 == 40 % 2.* [Video description ends]

This condition is exactly the same as the condition above, therefore, it evaluates to true. So far we've seen the relational operators in Python. Now let's move on to the logical operators.

[Video description begins] *In a cell, she enters the code, 5 > 3 and 4 < 6.* [Video description ends]

And as a logical operator that evaluates to true if both the operands on the either side of the and evaluate to true. The statement that you see on the screen, 5 > 3 and 4 < 6, will evaluate to true as both the expressions on the either side of the and are true.

[Video description begins] *In a cell, she enters the code, (5 > 3) and (4 < 6).* [Video description ends]

The condition that we just evaluated is equivalent to enclosing comparisons within a parenthesis.

[Video description begins] *She executes the code, (5 > 3) and (4 < 6). The output reads: True.* [Video description ends]

The and operator evaluates to true only if conditions on both the sides of the and are satisfied.

[Video description begins] *In a cell, she enters the code, 5 > 3 and 4 < 2.* [Video description ends]

5 > 3 is true, but 4 < 2 is false, therefore this expression evaluates to false.

[Video description begins] *In a cell, she enters the code, 10 == 10 == True.* [Video description ends]

Take a look at this expression 10 == 10 == true. You would expect this expression to evaluate to true, but this evaluates to false. Let us see why. Evaluation of conditions are from the left to the right. So the first evaluation will be is 10 == 10. This is true, but is 10 == true? That is false.

[Video description begins] *In a cell, she enters the code, 10 == 10 and 10 == True.* [Video description ends]

The statement above is as good as saying, 10 == 10 and 10 == true. Now you can clearly see why the statement above is wrong.

[Video description begins] *In a cell, she enters the code, (10 == 10) == True.* [Video description ends]

Then we evaluate 10 == 10 within parentheses and then compare it to true, the Boolean true, this expression evaluates to true. This is because 10 == 10 will produce true within parentheses, then true equal to the Boolean true is true. Let's take a look at another type of logical operator the logical or.

[Video description begins] *In a cell, she enters the code, 5 > 3 or 4 < 2.* [Video description ends]

The or evaluates to true if the condition on the either side of the or is true. The expression that you see on screen will return true. This is the because 5 > 3 evaluates to true. Even if one condition on the either side of the or operator returns true, the or condition will evaluate to true. Let's take a look at another expression, 5 > 3 or 4 < 6 here, both the conditions are true.

[Video description begins] *In a cell, she enters the code, 5 > 3 or 4 < 6.* [Video description ends]

So this expression will return true.

[Video description begins] *In a cell, she enters the code, 5 > 6 or 4 < 2.* [Video description ends]

5 > 6 or 4 < 2 here both the conditions are false, therefore, this expression evaluates to false. Let's take a look at the third type of the logical operator, the logical not.

[Video description begins] *In a cell, she enters the code, not(5 > 3 and 4 < 2).* [Video description ends]

When we use the not operator, it reverses the result. It returns true if the expression evaluates to false and vice versa. 5 > 3 and 4 < 2 should evaluate to false, notice the use of the not keyword. Now this expression will return true. Let's take a look at one more example, 5 > 3 or 4 < 6.

[Video description begins] *In a cell, she enters the code, not(5 > 3 and 4 < 6).* [Video description ends]

This expression should evaluate to true, but we're using the not keyword, so this will result in to false. In this video, we studied, comparison operators and logical operators. Comparison operators are used to compare two values. And logical operators are used to allow a program to make a decision based on multiple conditions.

If Statements with Primitive Datatypes

[Video description begins] *Topic title: If Statements with Primitive Datatypes. The presenter is Yukti Kalra. [Video description ends]*

Now that you've understood how the conditions work, we are now ready to step into some real programming.

[Video description begins]The IfStatement page opens in the jupyter notebook. [Video description ends]

Let's see how we can evaluate conditions using the if statements. Programming is all about making decisions based on the various conditions that are evaluated. If-else statements help us make those decisions and based on the conditions that we've evaluated, it will help us take those appropriate actions. Now we're ready for our first programming statement, the if statement.

[Video description begins] *In a cell, she enters the code, code starts: if 10 < 15: print("10 is less than 15"). Code ends.* [Video description ends]

The if statement that you see on the screen evaluates whether 10 is less than 15. If this condition evaluates to true, then we will print out 10 is less than 15. Now how does Python know? The print statement has to be executed only if the condition evaluates to true. Python understands this because of the way the indentation has been set up. Notice, the print statement has been indented within the if block. This indentations means that the print statement will be executed only if the condition is true. Notice the colon at the end of the if statement. This is extremely important. The colon signifies the start of the if block. If you try to write the if statement without a colon, it will throw an error. It will be a syntax error in Python.

[Video description begins] *She executes the aforementioned code. The output reads: 10 is less than 15.* [Video description ends]

The if statement evaluates to true and the print statement is executed.

[Video description begins] *In a cell, she enters the code, code starts: if 46 > 40 + 5: print("46 is greater than 45"). Code ends.* [Video description ends]

Let's evaluate the condition if 46 > 40 + 5 that is 45. If this condition evaluates to true, we want to print 46 is greater than 45 to the screen. The if condition evaluates to true, 46 is greater than 45 is printed out to the screen. Let's evaluate another condition.

[Video description begins] *In a cell, she enters the code, code starts: if 46 > 40: print("46 is greater than 40"). Code ends.* [Video description ends]

If 46 > 40, then print 46 is greater than 40. But notice the indentation of the print statement. The print statement is outside the if block. Let's see what happens if we execute this statement. Python throws an error, the error clearly states IndentationError. We need to have the print statement within the if block for this statement to execute without any errors.

[Video description begins] *In a cell, she enters the code, x_value = 65 y_value = 25.* [Video description ends]

Let's initialize two variables, x_value = 65 and y_value = 25. Let's evaluate a condition using these two variables in the if statement.

[Video description begins] *In a cell, she enters the code, code starts: if x_value < y_value: print('Value of x is less than the value of y'). Code ends.* [Video description ends]

If x value is less than y value, if this condition evaluates to true, then print value of x is less than the value of y. This condition evaluates to false which is why nothing gets printed out to the screen.

[Video description begins] *In a cell, she enters the code, code starts: if x_value > y_value: print('Value of x is greater than the value of y'). Code ends.* [Video description ends]

Let's flip the condition and check if the x_value > y_value.

[Video description begins] *She executes the aforementioned code.* [Video description ends]

Yes, it is indeed, Value of x is greater than the value of y is printed out to the screen.

[Video description begins] *In a cell, she enters the code, my_string = "Hello python world".* [Video description ends]

Let's initialize a variable, my_string with the string, Hello python world. Let's evaluate the condition to check if the string python is present in the my_string or not.

[Video description begins] *In a cell, she enters the code, code starts: if "python" in my_string: print('Yes, "python" is in my_string'). Code ends.* [Video description ends]

The in keyword allows you to check if the string Python is present in the my_string variable only. This condition evaluates to true, hence, Yes, python is in my_string is printed to the screen.

[Video description begins] *In a cell, she enters the code, code starts: if "pythonworld" in my_string: print('Yes, "python world" is in my_string'). Code ends.* [Video description ends]

Let's check if the string pythonworld is present in the my_string variable or not. Nothing gets printed out to the screen. This is because the python world present in the my_string variable contains a space. And the condition here is evaluating if the Python world without the space is present in the my_string or not.

[Video description begins] *In a cell, she enters the code, code starts: if "python world" in my_string: print('Yes, "python world" is in my_string'). Code ends.* [Video description ends]

Now let's check if python world is present in the my_string variable or not. This evaluates to true. Yes, python world is in my_string is printed out with the screen.

[Video description begins] *In a cell, she enters the code, code starts: if "Python World" in my_string: print('Yes, "python world" is in my_string'). Code ends.* [Video description ends]

The if statement that you see on the screen evaluates if python world with uppercase P and uppercase W are present in my_string or not. And nothing gets printed. That is because strings are case sensitive in Python.

[Video description begins] *In a cell, she enters the code, code starts: list_student = ["Ayden", "Gavin", "Ian", "Jose", "Jane"] list_student. Code ends. She executes it. The output reads: ['Ayden', 'Gavin', 'Ian', 'Jose', 'Jane'].* [Video description ends]

Here we have a list of students Ayden, Gavin, Ian, Jose, and Jane. Sometimes in programming you might want to check whether the student Jose is present in the list of students or not. Because if you try to access Jose without it being a part of this list, it might lead to an error condition.

[Video description begins] *In a cell, she types the code, code starts: if "Jose" in list_student: print("Jose is a student"). Code ends.* [Video description ends]

There's an easy way to do this in Python, if Jose in list_student, then print Jose as a student. The in keyword allows you to check whether an element is a part of a given list or not. This condition evaluates to true, Jose is a student, is printed out to the screen.

[Video description begins] *In a cell, she types the code, code starts: if "Eva" in list_student: print("Eva is a student"). Code ends.* [Video description ends]

Let's check whether the student Eva is present in our list or not, the if statement evaluates to false and nothing gets printed out to the screen.

If Statements with Complex Datatypes

[Video description begins] *Topic title: If Statements with Complex Datatypes. The presenter is Yukti Kalra. [Video description ends]*

[Video description begins] The IfStatement page opens in the jupyter notebook. [Video description ends]

Here, we have a tuple of teachers, Alice, Alexa, Robert, and Bella.

[Video description begins] *A cell contains the code, code starts: tuple_teachers = ("Alice", "Alexa", "Robert", "Bella") tuple_teachers. Code ends. She executes it. The output reads: ('Alice', 'Alexa', 'Robert', 'Bella').* [Video description ends]

Let's check if Alexa is present in the tuple or not. Execute this cell, Alexa is a teacher is printed out to the screen.

[Video description begins] *In a cell, she enters the code, code starts: if "Alexa" in tuple_teachers: print("Alexa is a teacher"). Code ends.* [Video description ends]

Let's see how the if statements work with dictionaries.

[Video description begins] *In a cell, she enters the code, code starts: student_score = {"Ayden":60, "Gavin":85, "Ian":76, "Jose":70, "Jane":82} student_score. Code ends.* [Video description ends]

We have a student_score dictionary over here where the keys are the student names and the values are the scores of those students.

[Video description begins] *She executes the aforementioned code. The output reads: {'Ayden': 60, 'Gavin': 85, 'Ian': 76, 'Jose': 70, 'Jane': 82}.* [Video description ends]

Let's check if the student Ian is present in the students score dictionary.

[Video description begins] *In a cell, she enters the code, code starts: if "Ian" in student_score: print("Ian is in the student_score dictionary"). Code ends.* [Video description ends]

This statement evaluates to true, and Ian is in the student_score dictionary is printed out to the screen. Let's check if the student Jane is present in the student_score dictionary. If this evaluates to true, we want to print out her score.

[Video description begins] *In a cell, she enters the code, code starts: if "Jane" in student_score: print("Jane's score:", student_score["Jane"]). Code ends.* [Video description ends]

We are accessing the score of the key Jane. This condition evaluates to true, Jane's score is 82.

[Video description begins] *The output reads: Jane's score: 82.* [Video description ends]

Let's check the score of another student.

[Video description begins] *In a cell, she enters the code, code starts: if "Ayden" in student_score: print("Ayden's score:", student_score["Ayden"]). Code ends.* [Video description ends]

If Ayden is present in the student_score dictionary, then print the score of Ayden. Ayden's score is 60.

[Video description begins] *She executes the aforementioned code. The output reads: Ayden's score: 60.* [Video description ends]

We have two variables here, a = 60, and b = 35.

[Video description begins] *In a cell, she enters the code, a = 60 b = 35*. [Video description ends]

Let's see how we can use the logical and in an if statement.

[Video description begins] *In a cell, she enters the code, code starts: if a > b and b < a: print("a is greater than b"). Code ends*. [Video description ends]

This if statement evaluates if a > b, and if b < a. If both these conditions are true, a is greater than b will be printed out to the screen, a is indeed greater than b and the if statement evaluates to true. Let's make a teeny little change in the condition above.

[Video description begins] *In a cell, she enters the code, code starts: if a > b and b > a: print("a is greater than b"). Code ends*. [Video description ends]

Now the if statement evaluate if a > b and b > a, we want to print a is greater than b out to the screen. And as you would have expected, nothing gets printed out to the screen.

[Video description begins] *In a cell, she enters the code, code starts: print(x_value) print(y_value). Code ends*. [Video description ends]

Let's check the values stored in the variables x_value and y_value that we had initialized earlier.

[Video description begins] *She executes the aforementioned code. The output reads: 65 25*. [Video description ends]

Now, we'll perform an addition operation with the x_value and the y_value and store it in the variable z_value.

[Video description begins] *In a cell, she enters the code, code starts: z_value = x_value + y_value z_value. Code ends. She executes it. The output reads: 90*. [Video description ends]

The z_value contains the value 90. Now let's evaluate a condition.

[Video description begins] *In a cell, she enters the code, code starts: if x_value > y_value and y_value < z_value: print("Both conditions are True"). Code ends*. [Video description ends]

If x_value > y_value and y_value < z_value, then print both conditions are true. As expected, the if statement evaluates to true.

[Video description begins] *In a cell, she enters the code, a,b. She executes this code. The output reads: (60, 35)*. [Video description ends]

Print out the values stored in the variables a and b that we had initialized earlier. Let's see how to use the logical or in an if statement. The if statement evaluates if a > b, or b < a.

[Video description begins] *In a cell, she enters the code, code starts: if a > b or b < a: print("At least one of the above conditions is true"). Code ends*. [Video description ends]

Then print, At least one of the above conditions is true. Even, if one of the conditions is true, the print statement will be executed. The condition a > b evaluates to true. The print statement, At least one of the above conditions is true, is printed out to the screen. We have a bike_price that's $716.

[Video description begins] *In a cell, she enters the code, bike_price = 716. [Video description ends]*

The result of a conditional evaluation is either true or false.

[Video description begins] In a cell, she enters the code, bike_is_electric = False. [Video description ends]

Which is why it should be no surprise to you that you can use Booleans when you use the if statements. Let's say we have a variable, bike_is_electric, set to False. We can use this variable in an if statement.

[Video description begins] *In a cell, she enters the code, code starts: if bike_price > 500 or bike_is_electric: print("At least one of the above conditions is true"). Code ends.* [Video description ends]

If the bike price is greater than $500 or bike_is_electric is set to true, then print at least one of the above conditions is true.

[Video description begins] *She executes the aforementioned code. The output reads: At least one of the above conditions is true.* [Video description ends]

Since, a logical or is being used here, the bike price is greater than $500 evaluates to true and the print statement is executed.

[Video description begins] *In a cell, she enters the code, code starts: if bike_price > 1000 or bike_is_electric: print("At least one of the above conditions is true"). Code ends.* [Video description ends]

Let's work with one more example, if the bike_price is more than $1000 or bike_is_electric, then the print statement will get executed. As you would have expected nothing gets printed out this clean, the bike_price > $1,000 and bike_ is_electric is set to False.

[Video description begins] *She points to the code, bike_price = 716 and bike_is_electric = False.* [Video description ends]

The if statement evaluates to False.

[Video description begins] *In a cell, she enters the code, code starts: if not bike_is_electric: print("It's a human-powered bike"). Code ends.* [Video description ends]

You can also use the logical not operator in your if statements, if not bike_is_electric, then print, It's a human-powered bike. Because bike_is_electric is set to False, not bike_is_electric == True. This condition evaluates to true. It's a human-powered bike is printed out to the screen.

[Video description begins] *In a cell, she enters the code, x = 45 y = 78.* [Video description ends]

Let's initialize x to be 45 and y to be 78.

[Video description begins] *In a cell, she enters the code, code starts: if not (x < y): print("x is not less than y"). Code ends.* [Video description ends]

Let's evaluate the condition if x not less than y, then print x is not less than y, but x is less than y.

[Video description begins] *She highlights the code, x = 45 y = 78.* [Video description ends]

The if statement evaluates to false because of the not keyword and nothing is printed out to the screen. Let's evaluate one more condition.

[Video description begins] *In a cell, she enters the code, code starts: if not (x > y): print("x is not greater than y"). Code ends.* [Video description ends]

If not x more than y, then print x is not greater than y. x greater than y should evaluate to false, but adding the not keyword returns true.

[Video description begins] *She executes the aforementioned code. The output reads: x is not greater than y.* [Video description ends]

The if statement evaluates to true and x is not greater than y is printed out to the screen.

If-else Elif Statements

[Video description begins] *Topic title: If-else Elif Statements. The presenter is Yukti Kalra. [Video description ends]*

Now that you've understood if statements, let's move on to the if-else statements.

[Video description begins] The IfElseAndElifStatement page opens in the jupyter notebook. [Video description ends]

The if statement can be paired with an else statement for branching into two conditions.

[Video description begins] *In a cell, she enters the code, code starts: if 10 > 20: print("10 is greater than 20") print("if block activated") else: print("10 is less than 20") print("else block activated"). Code ends.* [Video description ends]

The else clause catches anything that is not caught by the preceding conditions. When the if clause evaluates to false, the else clause is executed. Looking at a simple if-else statement could be the same as someone asking you a yes or no question. If 10 is greater than 20 and you want to take some action, that will be within the if block. If the if block evaluates to false, then you want to take some different set of actions. Those statements will be within the else block. If 10 is greater than 20, we will print 10 is greater than 20, if block activated. Else we'll print 10 is less than 20, else block activated. Notice the colon at the end of the else keyword. The colon signifies the start of the else clause and the indentation just below the else keyword signifies that the statements are part of the else block. And as you would expect, 10 is less than 20, else block activated is printed out to the screen. The if statement evaluates to false. The statements within the if block are ignored and the statements within the else block are executed.

[Video description begins] *She enters the code, bike_price = 7160 bike_price. She executes it. The output reads: 7160.* [Video description ends]

We have a motor bike priced at $7,160.

[Video description begins] *She enters the code, code starts: if bike_price <= 8000: print("It's a cheap bike") print("if block activated") else: print("It's an expensive bike") print("else block activated"). Code ends.* [Video description ends]

Let's check using the if-else statement whether this bike is a cheap bike or an expensive bike. The if condition evaluates if the bike price is less than or equal to $8,000, then it's a cheap bike, if block activated will be printed out to the screen. If this condition evaluates to false, then the else block will be executed. Since the bike price is less than $8,000, the if block is activated, and it's a cheap bike if block activated is printed out to the screen.

[Video description begins] *She enters the code, code starts: bike_price = 9000 if bike_price <= 8000: print("It's a cheap bike") else: print("It's an expensive bike"). Code ends.* [Video description ends]

Let's say we have a motorbike priced at $9,000. Let's check whether it's a cheap bike or an expensive bike. If the bike price is less than or equal to $8,000, it's a cheap bike, else it's an expensive bike. Notice the indentation of the else block. When you execute the cell, it will throw an error. The error clearly states it's a syntax error. The indentation of the else block here is wrong. The else block should start right below the if block.

[Video description begins] *She copies and pastes the aforementioned code in a cell and corrects its indentation.* [Video description ends]

Let's correct the indentation of the else block, execute the cell, the cell is executed without any errors. It's an expensive bike is printed out to the screen.

[Video description begins] *She enters the code, code starts: num = 50 print("num before expression: ", num) num = num - 20 if num > 20 else num + 20 print("num after expression: ", num). Code ends.* [Video description ends]

You can also write the if-else statement in a single line of code using the ternary operators. These operators are terse conditional expressions that test a condition and based on that evaluate a value. It allows to quickly test a condition instead of a multiline if statement. Oftentimes, it can be immensely helpful and can make your code compact but still maintaining. Python first evaluates the condition. If true, it evaluates the first expression. Otherwise, it evaluates the second. Let's take a look at the example on the screen. We have a variable num = 50 here.

[Video description begins] *She highlights the line of code, code starts: num = num - 20 if num > 20 else num + 20. Code ends.* [Video description ends]

We'll perform some operations on this num using the if-else conditional expressions. First, the Python interpreter will evaluate the condition, num > 20. If this evaluate to true, it will perform num - 20. And if this evaluates to false, it will perform num + 20. Since num is greater than 20, the if statement evaluates to true.

[Video description begins] *She executes the code, code starts: num = 50 print("num before expression: ", num) num = num - 20 if num > 20 else num + 20 print("num after expression: ", num). Code ends. The output reads: num before expression: 50 num after expression: 30.* [Video description ends]

And 20 is subtracted from num, the current value of num is 30. Let's look at the example above using the if-else statement with indentation. If num > 20, subtract 20 from num, else add 20 to num.

[Video description begins] *She enters the code, code starts: num = 50 print("num before expression: ", num) if num > 20: num = num - 20 else: num = num + 20 print("num after expression: ", num). Code ends.* [Video description ends]

And as expected num before the expression is 50, num after the expression is 30.

[Video description begins] *She executes the aforementioned code. The output reads: num before expression: 50 num after expression: 30.* [Video description ends]

Let's look at another example using the ternary operator. We have num set to 100.

[Video description begins] *She enters the code, code starts: num = 100 print("Number before expression : ", num) result = num / 5 if num < 50 else num * 5 print("Number after expression : ", result). Code ends.* [Video description ends]

We'll perform some operation on num and store it in the variable result. If num is less than 50, then divide num by 5, else multiply num by 5.

[Video description begins] *She executes the aforementioned code. The output reads: Number before expression : 100 Number after expression : 500.* [Video description ends]

And as you would expect, the else clause is executed, num is multiplied by 5. The result is 500. Now that you've understood the if-else statement, let's see what is an elif statement.

[Video description begins] *She enters the code, code starts: if 15 > 20: print("15 is greater than 20") print("if block activated") elif 15 < 20: print("15 is less than 20") print("elif block activated") else: print("Both are equal") print("else block activated"). Code ends.* [Video description ends]

We'll make the if statements progressively more complicated. Let's pass this code bit by bit. Let's start off with the first if statement. If 15 > 20, then print 15 is greater than 20 if block activated. If 15 < 20, then we'll move into the elif block. Elif is the short form for else plus if. The elif block evaluates if 15 < 20. If this evaluates to true, 15 < 20 elif block activated will be printed out to the screen. If the first two conditions evaluate to false, we'll move into the else block. Both are equal, else block activated will be printed out with the screen. As you can see, you can chain the if-else conditions together using the elif block. The most important thing to understand here is the conditions are evaluated in the order they're defined. First the if block is evaluated, then the elif block and then the else block. In this case, the elif block evaluates to true. The elif block is executed, 15 is less than 20, elif block activated is printed out to the screen.

[Video description begins] *She executes the aforementioned code. The output reads: 15 is less than 20 elif block activated.* [Video description ends]

Let's work with another example using the elif statement. Let's initialize variables a = 45 and b = 45.

[Video description begins] *She enters the code, code starts: a = 45 b = 45 if b > a: print("b is greater than a") print("if block activated") elif a == b: print("a and b are equal") print("elif block activated") else: print("a is greater than b") print("else block activated"). Code ends.* [Video description ends]

The if clause checks whether b is greater than a. The elif block checks whether a is equal to b. If the first two conditions evaluate to false, the else block will be executed. As you would have expected a and b are equal, elif block activated is printed out to the screen.

Nested If-else Statements

[Video description begins] *Topic title: Nested If-else Statements. The presenter is Yukti Kalra. [Video description ends]*

We have a motorbike priced at $20,000.

[Video description begins] The IfElseAndElifStatement page opens in the jupyter notebook. A cell contains the code, bike_price = 20000. [Video description ends]

Let's write code to check, whether bike is expensive or cheap.

[Video description begins] *In a cell, she enters the code, code starts: if bike_price < 6000: print("It's a cheap bike") elif bike_price >= 6000 and bike_price < 10000: print("It's a moderately priced bike") elif bike_price >= 10000 and bike_price < 15000: print("It's a somewhat expensive bike"). Code ends.* [Video description ends]

We have three categories here. If the bike price is less than $6,000, it's a cheap bike. If the first condition evaluates to false, we'll move into the elif block, which evaluates if the bike price is between $6,000 and $10,000. If this condition evaluates to true, it's a moderately priced bike. If the first two conditions evaluate to false, we'll move into the second elif block. Which evaluates if the bike price is between $10,000 and $15,000. Then it's a somewhat expensive bike. When you execute this cell, nothing is printed out to the screen, as none of the above conditions are met.

[Video description begins] *In a cell, she enters the code, code starts: if bike_price < 6000: print("It's a cheap bike") elif bike_price >= 6000 and bike_price < 10000: print("It's a moderately priced bike") elif bike_price >= 10000: print("It's a somewhat expensive bike") elif bike_price >= 20000: print("It's a very expensive bike"). Code ends.* [Video description ends]

Let's verify the fact that these conditions execute in the order they are defined. It's a cheap bike if the bike price is less than $6,000, it's a moderately priced bike if it is priced between $6,000 and $10,000. The second elif block evaluates, if the bike price is more than or equal to $10,000, it's a somewhat expensive bike. And the third elif block evaluates if the bike price is more than or equal to $20,000, it's a very expensive bike. The bike is priced at $20,000, so the second elif block and the third elif block both evaluate to true. For this example, when you execute this cell, it's a somewhat expensive bike is printed out to the screen. We see only one print statement executed. This is because the second elif statement evaluates to true. The Python interpreter does not evaluate the third elif statement, once the second elif statement evaluates to true.

[Video description begins] *In a cell, she enters the code, code starts: x = 25 y = 35 z = 45. Code ends.* [Video description ends]

We have three variables here, x = 25, y = 35, and z = 45. There might be a situation when you want to evaluate another condition after a condition resolves to true.

[Video description begins] *In a cell, she enters the code, code starts: if x < y: print("The first condition is true") if x < z: print("Both conditions are true") else: print("The first condition is true, the second one is false"). Code ends.* [Video description ends]

In such a situation, you can use a nested if-else construct. We can place an if statement within another if statement. Nesting control is statements makes us check multiple conditions. Let's take a look at an example of nested if-else statements. If x < y, then print the first condition is true. Notice the indentation of the second if and the else block. The second if and the else block are part of the first if block. If the first if block evaluates to true, we will enter the first if block. And then the Python interpreter will go on to evaluate if x < z. If this evaluates to true, it will print both conditions are true. Else, it will print the first condition is true, the second one is false. Let's execute this cell. The first condition is true. Both conditions are true is printed out to the screen. The first if condition evaluates to true. So we enter the first if block, we print the first statement, the first condition is true. Then we check if x < z, this also evaluates to true. So we print both conditions are true. Let's take a look at another example of nested if statements.

[Video description begins] *In a cell, she enters the code, code starts: x = 55 if x < y: print("The first condition is true") if x < z: print("Both conditions are true") else: print("The first condition is true, the second one is false") else: print("The first condition is false"). Code ends.* [Video description ends]

We have x set to 55 here. We'll first evaluate if x < y. If this evaluates to true, we'll move into the first if block and print the first condition is true. Then the Python interpreter will evaluate the nested if block. If x < z, then Both conditions are true will be printed out to the screen. If the nested if block evaluates to false, we'll move into the nested else block. The else clause that you see right below the second if clause. The first condition is true, the second one is false will be printed out to the screen. If the first if evaluates to false, then we will move into the second else block. The else block which is right below the first if. y is 35, and x is 55.

[Video description begins] *She executes the aforementioned code. The output reads: The first condition is false.* [Video description ends]

So the first if block evaluates to false. So all the statements within the first if block are bypassed, and the else is executed. The first condition is false is printed out to the screen. Let's take a look at another example. We'll accept an input from the user.

[Video description begins] *In a cell, she enters the code, code starts: age = int(input("Enter your age: ")) if age >= 15: if age >= 20: print("You are too old for this camping trip!") else: print("You are of the right age for this camping trip!") else: print("You are too young for this camping trip!"). Code ends.* [Video description ends]

The user should enter his or her age. We'll check, whether the person can go for a camping trip or not. The age limit for this particular camping trip is from 15 to 20. We'll use the nested if-else statements to check whether a person is eligible for this camping trip or not. If the age of the user is more than equal to 15, then we'll enter the first if block. If the first if block evaluates to false. That is the age of the person is less than 15, we'll move into the else block, which will print out, You are too young for this camping trip. If the age is more than equal to 15, then we will check if the age of the user is more than 20. If the nested if, the second if statement, evaluates to false, we will move onto the nested else statement. Which will print out, You are of the right age for this camping trip. Execute this cell, the age entered is 21.

[Video description begins] *The output reads: Enter your age: 21 You are too old for this camping trip!*. [Video description ends]

First, we'll evaluate the first if statement. If age is more than or equal to 15, this evaluates to true. Then we'll evaluate the nested if statement, the second if. If age is more than 20, the nested if block evaluates to true. You are too old for this camping trip is printed out to the screen. Let's check the eligibility of another user. The age entered is 18.

[Video description begins] *The output reads: Enter your age: 18 You are of the right age for this camping trip!*. [Video description ends]

The first if block, if age is more than equal to 15 evaluates to true. We move into to the first if block and evaluate the first nested if statement, if age is more than 20. This evaluates to false, so we move into the nested else block. You are of the right age for this camping trip is printed out to the screen. Let's look at another real-world example. We want to calculate the shipping cost incurred based on various conditions.

[Video description begins] *In a cell, she enters the code, code starts: total = int(input("What is the total amount of your online shopping? ")) country = input("USA or Canada? ") if country == "USA": if total <= 50: print ("Shipping Cost is $9.00") elif total <= 100: print ("Shipping cost is $6.00") else: print ("Shipping is FREE") if country == "Canada": if total <= 50: print ("Shipping Cost is $12.00") elif total <= 100: print ("Shipping Cost is $8.00") else: print (" Shipping is FREE"). Code ends*. [Video description ends]

Such as, the total amount of the online shopping, and the country where the order has to be delivered.

[Video description begins] *She highlights the line of code, code starts: total = int(input("What is the total amount of your online shopping? ")). Code ends*. [Video description ends]

We have a total variable here which accepts an input from the user. It stores the total amount of online shopping done by the user.

[Video description begins] *She highlights the line of code, code starts: country = input("USA or Canada? "). Code ends*. [Video description ends]

The country variable accepts an input from the user. It stores whether the country is USA or Canada. Execute this cell.

[Video description begins] *The output reads: What is the total amount of your online shopping? 80 USA or Canada? USA Shipping Cost is $6.00*. [Video description ends]

The total amount of online shopping is $80, and the country is USA. The first if block evaluates if the country is USA, it evaluates to true. So we enter the first if block. Now we'll evaluate the first nested if block, if total is less than or equal to $50. This evaluates to false, so we'll move to the elif block. The nested elif block evaluates if the total is less than or equal to $100, this evaluates to true. So we enter the nested elif block. Shipping cost is $6 is printed out to the screen. Let's check the shipping cost for another user.

[Video description begins] *In the output, she alters the values 80 to 40 and USA to Canada. The output reads: What is the total amount of your online shopping? 40 USA or Canada? Canada Shipping Cost is $12.00*. [Video description ends]

This user has shopped for $40 and belongs to Canada. Python checks for the first outer if block, if country is equal to USA, which evaluates to false. So we move on to the second outer if block, which is, if country is equal to Canada, this evaluates to true. Now, we will enter second outer if block, then we'll evaluate the nested if clause, if total is less than or equal to $50. This evaluates to true, shipping cost is $12 for this order.

If-else Statements with Complex Datatypes

[Video description begins] *Topic title: If-else Statements with Complex Datatypes. The presenter is Yukti Kalra.* [Video description ends]

Let's see how we can use the if-else statements with complex data types.

[Video description begins] *The IfElseStatementInComplexDatatypes page opens in the jupyter notebook.* [Video description ends]

Instantiate a list of fruits with apple, orange, grape, banana, and avocado.

[Video description begins] *In a cell, she enters the code, code starts: fruit_list = ["apple", "orange", "grape", "banana", "avocado"]. Code ends.* [Video description ends]

There are some special checks that we can perform with list using the if-else statements.

[Video description begins] *In a cell, she enters the code, code starts: if "grape" in fruit_list: print("Yes, grape is in the fruit list") else: print("No, grape is not in the fruit list"). Code ends.* [Video description ends]

Sometimes in a program, you might want to check if the fruit grape is present in the list of fruits. Because, if you try to access grape without it being a part of this list, it may lead to an error condition. There's an easy way to do this in Python. If grape in fruit_list, notice the in keyword. The in statement allows you to check whether an element is present in the list or not. The if block evaluates to true. Yes, grape is in the fruit list, is printed out to screen. Let's check for other elements in this fruit_list.

[Video description begins] *In a cell, she enters the code, code starts: if "strawberry" in fruit_list: print("strawberry is in the fruit list") elif "orange" in fruit_list: print("orange is in the fruit list") else: print("Both strawberry and orange are not in the fruit list"). Code ends.* [Video description ends]

If strawberry in fruit_list, print, strawberry is in the fruit list. If the first if block evaluates to false, then check if orange is present in the fruit_list. If the elif clause evaluates to true, then print, orange is in the fruit list. If the first two conditions evaluate to false, then print Both strawberry and orange are not in the fruit list. And as you would expect, orange is in the fruit list is printed out to screen. You can also check whether an element is present at a particular index in the list using if-else statements.

[Video description begins] *In a cell, she enters the code, code starts: if fruit_list[1] == "orange": print("True") else: print("False"). Code ends.* [Video description ends]

We want to check if the fruit orange is present at index 1 of the fruit_list.

[Video description begins] *She highlights the line of code, if fruit_list[1] == "orange":.* [Video description ends]

If this condition evaluates to true, we'll print True, else we'll print False. The fruit orange is present at index 1 of the fruit_list, so True is printed out to screen.

[Video description begins] *In a cell, she enters the code, code starts: if fruit_list[4] == "avocado": print("Yes, avocado is at the fourth index") print("Replacing avocado with strawberry at the fourth index") fruit_list[4] = "strawberry" print(fruit_list) else: print("avocado is not at the fourth index"). Code ends.* [Video description ends]

We can use if-else statements to update an element of a list.

[Video description begins] *She highlights the line of code, if fruit_list[4] == "avocado":.* [Video description ends]

First, we'll check if the fruit avocado is present at index 4 of the fruit_list.

[Video description begins] *She highlights the code: fruit_list[4] = "strawberry" print(fruit_list).* [Video description ends]

If this evaluates to true, we'll update the element at index 4 to be strawberry.

[Video description begins] *She highlights the code, code starts: else: print("avocado is not at the fourth index"). Code ends.* [Video description ends]

Else we'll simply print, avocado is not at the fourth index. The if block evaluates to true, avocado is present at index 4 of the list.

[Video description begins] *She executes the code, code starts: if fruit_list[4] == "avocado": print("Yes, avocado is at the fourth index") print("Replacing avocado with strawberry at the fourth index") fruit_list[4] = "strawberry" print(fruit_list) else: print("avocado is not at the fourth index"). Code ends. The output reads: Yes, avocado is at the fourth index Replacing avocado with strawberry at the fourth index ['apple', 'orange', 'grape', 'banana', 'strawberry'].* [Video description ends]

We'll replace this element with the fruit strawberry, print the fruit_list, and you can see strawberry at index 4. Let's see how if-else statements work with tuples. We have a tuple of cars, Toyota Camry, Honda Accord, Honda Civic, and Toyota Corolla.

[Video description begins] *In a cell, she enters the code, code starts: car_tuple = ('Toyota Camry', 'Honda Accord', 'Honda Civic', 'Toyota Corolla'). Code ends.* [Video description ends]

We can check whether an element is present in the tuple or not.

[Video description begins] *In a cell, she enters the code, code starts: if "Honda Accord" in car_tuple: print("Honda Accord is present in our car tuple") else: print("Honda Accord is not present in our car tuple"). Code ends.* [Video description ends]

We'll use the in keyword, if Honda Accord in car_tuple. If this evaluates to true, we'll print Honda Accord is present in our car tuple, else, Honda Accord is not present in our car tuple. And as you would expect, Honda Accord is present in our car tuple is printed out to screen.

[Video description begins] *In a cell, she enters the code, code starts: if "Ducati Monster" in car_tuple: print("Ducati Monster is a car") else: print("Ducati Monster is not a car"). Code ends.* [Video description ends]

Let's see whether the element Ducati Monster is present in the car_tuple or not. If Ducati Monster in car_tuple print Ducati Monster is a car, else Ducati Monster is not a car.

[Video description begins] *In the code, code starts: car_tuple = ('Toyota Camry', 'Honda Accord', 'Honda Civic', 'Toyota Corolla'), code ends, she highlights, code starts: ('Toyota Camry', 'Honda Accord', 'Honda Civic', 'Toyota Corolla'). Code ends.* [Video description ends]

As you can see in the tuple of cars, Ducati Monster is not present, the if block evaluates to false, Ducati Monster is not a car is printed out to screen. You can check for the presence of more than one element in a tuple, by using the and keyword.

[Video description begins] *In a cell, she enters the code, code starts: if "Ducati Monster" in car_tuple and 'Honda Accord' in car_tuple: print("Ducati Monster and Honda Accord are both cars") else: print("At least one of Ducati Monster and Honda Accord is not a car"). Code ends.* [Video description ends]

We want to check whether Ducati Monster and Honda Accord are present in the car tuple or not. If this evaluates to true, Ducati Monster and Honda Accord are both cars, will be printed out to screen.

[Video description begins] *She executes the aforementioned code. The output reads: At least one of Ducati Monster and Honda Accord is not a car.* [Video description ends]

Since Ducati Monster is not present in the tuple of cars, the if block evaluates to false, so we move into the else block. At least one of Ducati Monster and Honda Accord is not a car is printed out to screen.

[Video description begins] *In a cell, she enters the code, code starts: if "Ducati Monster" in car_tuple or 'Honda Accord' in car_tuple: print("At least one of Ducati Monster and Honda Accord is a car") else: print("Neither Ducati Monster nor Honda Accord is a car"). Code ends.* [Video description ends]

We can also use the logical or operator and check the presence of elements in a tuple. If Ducati Monster or Honda Accord, one of these elements is present in the car_tuple, the if condition will evaluate to true, Honda Accord is present in the car_tuple. The if block has executed At least one of the Ducati Monster and Honda Accord is a car is printed out the screen.

[Video description begins] *In a cell, she enters the code, code starts: salary_details = {"Lisa":25000, "Jason":45000, "Cooper":35000, "Elias":23000, "Jordan":77000}. Code ends.* [Video description ends]

Here we have a dictionary of salaries of Lisa, Jason, Cooper, Elias, and Jordan print out this dictionary.

[Video description begins] *In a cell, she enters the code: salary_details. The output reads: {'Lisa': 25000, 'Jason': 45000, 'Cooper': 35000, 'Elias': 23000, 'Jordan': 77000}.* [Video description ends]

Let's check whether the key Lisa is present in the dictionary of salaries using if-else statements.

[Video description begins] *In a cell, she enters the code, code starts: if "Lisa" in salary_details: print("We have the salary details for Lisa") else: print("We don't have the salary details for Lisa"). Code ends.* [Video description ends]

If Lisa in salary_details will automatically look at the keys of this dictionary, and check whether Lisa is a valid key.

[Video description begins] *She executes the aforementioned code. The output reads: We have the salary details for Lisa.* [Video description ends]

In this case, the key Lisa is present in the dictionary of salaries, the if statement evaluates to true, the print statement within the if block is executed.

[Video description begins] *In a cell, she enters the code, code starts: if "Ruby" in salary_details: print("We have the salary details for Ruby ") else: print("We don't have the salary details for Ruby "). Code ends.* [Video description ends]

Let's check whether the employee Ruby is present in our dictionary or not. The if statement evaluates to false, We don't have the salary details for Ruby is printed out to screen.

[Video description begins] *In a cell, she enters the code, code starts: if "Cora" in salary_ details: print("We have salary details for Cora") else : salary_details["Cora"] = 31000 print("Cora's annual income is %s"%salary_details["Cora"]). Code ends.* [Video description ends]

In addition to checking whether a key is present in the dictionary or not, we can also update new keys to the dictionary using if-else statements.

[Video description begins] *She highlights the code, code starts: if "Cora" in salary_ details: print("We have salary details for Cora"). Code ends.* [Video description ends]

First, let's verify if the key Cora is present in the dictionary of salaries.

[Video description begins] *She highlights the line of code, salary_details["Cora"] = 31000.* [Video description ends]

If this evaluates to false, that is Cora is not present in the dictionary, then we'll update the dictionary of salaries with the key Cora, and we'll update the corresponding salary to be $31,000.

[Video description begins] *She executes the code, code starts: if "Cora" in salary_ details: print("We have salary details for Cora") else : salary_details["Cora"] = 31000 print("Cora's annual income is %s"%salary_details["Cora"]). Code ends. The output reads: Cora's annual income is 31000.* [Video description ends]

Since the key Cora is not present in our dictionary, the if block evaluates to false. We move into the else block, we update the dictionary with Cora, Cora's annual income is $31,000.

[Video description begins] *In a cell, she enters the code, salary_details. She executes it. The output reads: {'Lisa': 25000, 'Jason': 45000, 'Cooper': 35000, 'Elias': 23000, 'Jordon': 77000, 'Cora': 31000}.* [Video description ends]

Print out the dictionary and you can see, the key Cora has been updated to this dictionary.

[Video description begins] *In the output, she highlights Cora': 31000.* [Video description ends]

Let's look at another example.

[Video description begins] *In a cell, she enters the code, code starts: age_details = {"Lisa": 25, "Jason": 30, "Cooper": 29, "Sarah": 22}. Code ends.* [Video description ends]

We have a dictionary here, with the age_details of Lisa, Jason, Cooper, and Sarah.

[Video description begins] *In a cell, she enters the code, code starts: if age_details["Lisa"] < age_details["Jason"]: print("Jason is older than Lisa") if age_details["Jason"] > age_details["Cooper"]: print("Jason is older than Cooper") if age_details["Cooper"] < age_details["Sarah"]: print("Cooper is younger than Sarah") elif age_details["Cooper"] > age_details["Sarah"]: print("Jason is the oldest person in the given dictionary") else: print("Jason is not the oldest person in the given dictionary"). Code ends.* [Video description ends]

Here we have nested if-else statements. There's a lot going on here, but do not worry, let's pass this bit by bit. The objective of this problem is to find whether Jason is the oldest person in the given dictionary or not. First, we will check if the age of Lisa is less than that of Jason, so we access the values of the keys, Lisa and Jason. If this condition evaluates to false, we will move into the else block and print Jason is not the oldest person in the given dictionary. If the first if block evaluates to true we will print, Jason is older than Lisa. Then the nested if will evaluate if the age of Jason is greater than age of Cooper. If this evaluates to true, we'll print, Jason is older than Cooper. And we will move into the second nested if, which will check if the age of Cooper is less than age of Sarah. If this evaluates to true, we will print Cooper is younger than Sarah. Otherwise we'll move into the elif block, the nested elif block, which checks if the age of Cooper is more than age of Sarah. If this evaluates to true, Jason is the oldest person in the given dictionary, is printed out to screen.

[Video description begins] *The output reads: Jason is older than Lisa Jason is older than Cooper Jason is the oldest person in the given dictionary.* [Video description ends]

Execute this cell, Jason is indeed the oldest person in the given dictionary.

[Video description begins] *In a cell, she enters the code, code starts: details = [['Jane', 'Amanda', 'Emma'], [35, 40, 50], [20000, 50000, 40000]]. Code ends. [Video description ends]*

Let's work with a nested list, the list has three inner lists.

[Video description begins]In the code, she highlights, [['Jane', 'Amanda', 'Emma'],. [Video description ends]

The first inner list is the list of names.

[Video description begins] *In the code, she highlights, [35, 40, 50].* [Video description ends]

The second inner list is the list of their respective ages.

[Video description begins] *In the code, she highlights, [20000, 50000, 40000]].* [Video description ends]

The third inner list is their respective salaries.

[Video description begins] *He enters the code, code starts: max_sal = max(details[2]) if (details[2][1] == max_sal): if(details[1][1] > 30): print(details[0][1], "has the highest salary and her age is greater than 30") elif(details[1][1] == 30): print(details[0][1],"has the highest salary and she is 30 years old") else: print("Amanda has the highest salary and her age is less than 30") else: print("Amanda is not the highest paid employee"). Code ends.* [Video description ends]

Let's write code to check who is the highest paid employee, and what is the age bracket that she falls into?

[Video description begins] *She highlights the line of code, max_sal = max(details[2]).* [Video description ends]

Let's store the maximum salary in the variable max_sal.

[Video description begins] *She highlights the line of code, if (details[2][1] == max_sal):. In the code, code starts: details = [['Jane', 'Amanda', 'Emma'], [35, 40, 50], [20000, 50000, 40000]],. Code ends. She points to 50000.* [Video description ends]

We want the maximum value of the inner list at index 2 of the outer list. Let's check whether Amanda with salary of $50,000 has the maximum salary or not. If this statement evaluates to true, we will move into the if block. Then we'll check her age bracket using if, elif, and else statements. We access the age of Amanda by details[1][1]. We'll check if the age is more than 30, if this evaluates to false, we'll move in to the elif block to check if the age is equal to 30. If both if and elif blocks evaluate to false, we'll move in to the else block and print Amanda has the highest salary and her age is less than 30. Execute this cell, Amanda has the highest salary and her age is greater than 30.

Type Conversions with Primitive Datatypes

[Video description begins] *Topic title: Type Conversions with Primitive Datatypes. The presenter is Yukti Kalra. [Video description ends]*

We work with a number of different data types in Python.

[Video description begins]The TypeConversion-01 page opens in the jupyter notebook. [Video description ends]

Primitive data types such as integers, Booleans, strings, and so on, and also complex data types such as lists, tuples, dictionaries, etc. Python gives you built-in ways to convert between types. Such as strings can be converted to numbers, numbers can be converted to strings, and lists can be converted to tuples and so on. Let's start off by looking at an inbuilt function, the type function.

[Video description begins] *In a cell, she enters the code, type(66).* [Video description ends]

The type function is mostly used for debugging purposes. The type method returns the class type of the argument passed as a parameter. Let's examine the type of the value 66, it's an integer.

[Video description begins] *She executes the aforementioned code. The output reads: int*. [Video description ends]

This integer can be converted into a floating point.

[Video description begins] *In a cell, she enters the code, float(66)*. [Video description ends]

This can be done by using the float inbuilt function in Python.

[Video description begins] *She executes the aforementioned code. The output reads: 66.0*. [Video description ends]

Notice the way it has been printed out to screen, there's a .0 indicating that it's a float.

[Video description begins] *In a cell, she enters the code, a = 66 type(a)*. [Video description ends]

Let's initialize a variable with the value 66.

[Video description begins] *She executes the aforementioned code. The output reads: int. [Video description ends]*

We'll check the type of a, the variable a is of type int.

[Video description begins] In a cell, she enters the code, my_float = float(a) my_float. [Video description ends]

Let's convert the variable a to a floating point by invoking the built-in function float and storing it in the variable my_float. Print out my_float to screen and you can see it's a float.

[Video description begins] *She executes the aforementioned code. The output reads: 66.0*. [Video description ends]

Let's verify this by checking the type of the variable my_float.

[Video description begins] *In a cell, she enters the code, type(my_float).[Video description ends]*

It is indeed a float.

[Video description begins] She executes the aforementioned code. The output reads: float. [Video description ends]

Let's check the type of the value 390.8.

[Video description begins] *In a cell, she enters the code, type(390.8). She executes it. The output reads: float*. [Video description ends]

It is a float. What if we want to convert this float to an integer?

[Video description begins] *In a cell, she enters the code, int(390.8)*. [Video description ends]

We will use the built-in function int and we get the value 390. The built-in function int rounds down the value of float.

[Video description begins] *In a cell, she enters the code, code starts: b = 66.6 my_int = int(b) my_int. Code ends*. [Video description ends]

Let's initialize a variable b with the value 66.6.

[Video description begins] *She highlights the line of code, my_int = int(b)*. [Video description ends]

Now, we'll convert this float into an integer by invoking the built-in function int.

[Video description begins] *She executes the code, code starts: b = 66.6 my_int = int(b) my_int. Code ends. The output reads: 66.* [Video description ends]

Print the my_int variable and you can see that it has been converted into an integer, the value 66. The value after the decimal point has been truncated.

[Video description begins] *In a cell, she enters the code, type(b). She executes it. The output reads: float. [Video description ends]*

Let's verify the type of the variable b, it is a float.

[Video description begins]In a cell, she enters the code, type(my_int). [Video description ends]

Also verify the type of the variable my_int, it is an int.

[Video description begins] *She executes the aforementioned code. The output reads: int. [Video description ends]*

I will now initialize two variables, num_int with the value 76, and num_float with the value 23.4.

[Video description begins]In a cell, she enters the code, code starts: num_int = 76 num_float = 23.4 print("datatype of num_int:", type(num_int)) print("datatype of num_float:", type(num_float)). Code ends. [Video description ends]

Print out the type of the variable num_int and type of the variable num_float.

[Video description begins] *She executes the aforementioned code. The output reads: datatype of num_int: <class 'int'> datatype of num_float: <class 'float'>.* [Video description ends]

Datatype of the variable num_int belongs to the class int, and datatype of the variable num_float belongs to the class float.

[Video description begins] *In a cell, she enters the code, code starts: new_num = num_int + num_float print(new_num) print("datatype of new_num is", type(new_num)). Code ends.* [Video description ends]

Let's perform an addition operation on variables num_int and num_float. I'm going to store the result in the variable new_num.

[Video description begins] *She executes the aforementioned code. The output reads: 99.4 datatype of new_num is <class 'float'>.* [Video description ends]

The variable new_num holds the value 99.4, and the datatype of the variable new_num is a float. The addition operation between an integer and a float results in a float.

[Video description begins] *In a cell, she enters the code, my_float = 66.999999.* [Video description ends]

Here we have a variable, my_float, set to a decimal value. Let's convert my_float into an integer and store it in the variable my_num.

[Video description begins] *In a cell, she enters the code, code starts: my_num = int(my_float) my_num. Code ends. She executes it. The output reads: 66.* [Video description ends]

The value stored in the variable my_num is 66. The values after the decimal point have been truncated. Let's look at another example, a = 125.0, b = 390.8.

[Video description begins] *In a cell, she enters the code, code starts: a = 125.0 b = 390.8 print(int (a)) print(int (b)). Code ends.* [Video description ends]

Invoke the built-in function int on a and b, and we get integer values 125 and 390.

[Video description begins] *In a cell, she enters the code, int('12').* [Video description ends]

Let's see what happens when we invoke the built-in function int on a string. We know it's a string because it is enclosed within single quotes. Execute this cell and we get an integer 12. We know it's an integer because there are no quotes around it. Converting the string 12 to an integer is a valid conversion.

[Video description begins] *In a cell, she enters the code, a = "66" type(a).* [Video description ends]

We have a variable a here set to the string 66.

[Video description begins] *She executes the aforementioned code. The output reads: str.* [Video description ends]

The type of the variable a is a string.

[Video description begins] *In a cell, she enters the code, my_int = int(a) my_int.* [Video description ends]

We invoke the int function on the variable a and we'll store the result in the variable my_int.

[Video description begins] *She executes the aforementioned code. The output reads: 66.* [Video description ends]

The string 66 has been converted to the integer 66. You can verify this by checking the type of the variable my_int.

[Video description begins] *In a cell, she enters the code, type (my_int). She executes it. The output reads: int.* [Video description ends]

It is indeed an integer.

[Video description begins] *In a cell, she enters the code, code starts: num_int = 123 num_str = "456" print(num_int + num_str). Code ends.* [Video description ends]

Initialize two variables, num_int with the value 123 and num_str with the string 456. Perform the addition operation on variables num_int and num_str. However, Python throws an error here. Python does not support the addition of an integer and a string. Verify types of variables num_int and num_str.

[Video description begins] *In a cell, she enters the code, code starts: print("Data type of num_int:", type(num_int)) print("Data type of num_str:", type(num_str)). Code ends. She executes it. The output reads: Data type of num_int: <class 'int'> Data type of num_str: <class 'str'>.* [Video description ends]

Num_int is an integer and num_str is a string.

[Video description begins] *In a cell, she enters the code, code starts: num_str = int(num_str) print("Data type of num_str:", type(num_str)). Code ends.* [Video description ends]

Let's convert the num_string variable from a string to an int.

[Video description begins] *She executes the aforementioned code. The output reads: Data type of num_str: <class 'int'>.* [Video description ends]

Let's verify the type of the variable num_string, after the conversion, it is an integer. Once again, let's perform an addition operation on variables num_int and num_str.

[Video description begins] *In a cell, she enters the code, code starts: sum_value = num_int + num_str print(sum_value) print("Data type of sum_value:",type(sum_value)). Code ends.* [Video description ends]

Store the result in the variable sum_value. Now, this will not show any error because both the variables are integers. The addition results in the value 579 and the data type of some_value is int.

[Video description begins] *She executes the aforementioned code. The output reads: 579 Data type of sum_value: <class 'int'>.* [Video description ends]

What will happen if we try to convert the word five into an integer?

[Video description begins] *In a cell, she enters the code, code starts: value_str = "five" value_int = int(value_str). Code ends.* [Video description ends]

If this string were a number 5 instead of the word five, this conversion would have been possible. However, in this case, Python throws an error. This is because Python does not know the word five means the number 5. The error clearly states invalid literal for int() with base 10. The error is a bit cryptic here but it is pretty clear that what you've passed into the int function is invalid. The reason that this error refers to the base 10, because you can have your integers expressed as hexadecimal, base 16, binary format, base 2, octal format, base 8, and so on.

[Video description begins] *In the line of code, value_str = "five", she highlights "five".* [Video description ends]

This particular string that we passed in was not a valid integer with the base 10 format. Let's see if you can convert an integer into a string.

[Video description begins] *In a cell, she enters the code, str(12).* [Video description ends]

We invoke the built-in function string on the number 12, this results in a string.

[Video description begins] *She executes the aforementioned code. The output reads: '12'.* [Video description ends]

We know it's a string because it is enclosed within quotes.

[Video description begins] *In a cell, she enters the code, code starts: my_str = str(num_int) my_str. Code ends.* [Video description ends]

Let's convert the variable num_int into a string and store the result in the variable my_str.

[Video description begins] *She executes the aforementioned code. The output reads: '123'.* [Video description ends]

The variable my_str now has 123 string value. Let's look at a use case when you might need to convert your float or integer value to a string.

[Video description begins] *In a cell, she enters the code, code starts: my_float = 5524.53 print("John has " + str(my_float) + " points."). Code ends.* [Video description ends]

While programming, we have a variable my_float initialized with the value 5,524.53. We want to print the statement that John has 5,524.53 points. Addition works differently with strings than with floats. With integers and floats, the addition operation calculates the sum of all values.

[Video description begins] *She executes the aforementioned code. The output reads: John has 5524.53 points.* [Video description ends]

Whereas with strings, addition operation concatenates strings to combine it into a one big string. It is necessary to convert the float value into a string because an addition operation on a string and a float throws an error. Let's verify this fact by looking at another example.

[Video description begins] *In a cell, she enters the code, code starts: user = "John" lines = 50 print("Congratulations, " + user + "! You just wrote " + lines + " lines of code."). Code ends.* [Video description ends]

We have a variable user set to John and a variable lines set to the integer 50. Let's see what will happen when we execute this print statement. This will throw an error, and the error clearly states it can only concatenate str to str, not int. Let's debug the error above.

[Video description begins] *In a cell, she enters the code: code starts: user = "John" lines = 50 print("Congratulations, " + user + "! You just wrote " + str (lines) + " lines of code."). Code ends.* [Video description ends]

Notice now we're converting the lines variable to a string. Now when we'll execute the cell, Congratulations, John! You just wrote 50 lines of code is printed out to screen. Let's take a look at another use case.

[Video description begins] *In a cell, she enters the code, code starts: number = input() plus_ten = number + 10. Code ends.* [Video description ends]

We'll accept an input from the user and store it in the variable number, then we'll perform an addition operation and store it in the variable plus_ten. Let's see what happens when we execute this cell. We enter the number 6, and it throws an error. The error states TypeError: can only concatenate str (not “int”) to str. By default, the input function results in a string. So when we try to add the number variable to the integer 10, we get an error. Verify the type of the number variable. Let's debug the TypeError.

[Video description begins] *In a cell, she enters the code, type(number). She executes it. The output reads: str. [Video description ends]*

[Video description begins] In a cell, she enters the code, code starts: number = int(input()) plus_ten = number + 10 print ("If we add 10 to this number, we get " + plus_ten). Code ends. [Video description ends]

We'll pass the input function into the built-in function int, then we'll add 10 to the number variable and store it in the plus_ten variable. Execute this cell, we enter the number 7, and we still get a TypeError. Now this error is because of the print statement. We should be very careful while debugging errors. We did not make the required changes in the print statement. The plus_ten variable is an integer and we're trying to add it to a string.

[Video description begins] *In a cell, she enters the code, code starts: number = int(input()) plus_ten = number + 10 print ("If we add 10 to this number, we get " + str(plus_ten)). Code ends.* [Video description ends]

Let's debug this error as well, notice the print statement will convert the plus_ten variable into a string. Execute this cell, it will not throw any errors. We enter the number 6, If we add 10 to this number, we get 16 is printed out to screen. Let's take a look at another example.

[Video description begins] *In a cell, she enters the code: code starts: number = int(input()) plus_ten = number + 10 plus_twenty = number + 20 print ("\nIf we add 10 and 20 to this number, we get %s and %s " %(plus_ten, plus_twenty)). Code ends.* [Video description ends]

We'll accept an input from the user and store it in the variable number. Notice we're invoking the int function on the input function. We'll perform two operations on this number. We'll add the integer 10 to this number and store it in the variable plus_ten, and then we will add 20 to this number and store it in the variable plus_twenty. Notice the print statement. Instead of converting variables plus_ten and plus_twenty into string, we can use string formatting. Notice the use of %s in the print statement and the use of % symbol right before the tuple of plus_ten, plus_twenty. This is another way of concatenating integers with strings.

[Video description begins] *She executes the aforementioned code and enters the number 67. The output reads: 67 If we add 10 and 20 to this number, we get 77 and 87.* [Video description ends]

Type Conversions with Complex Datatypes

[Video description begins] *Topic title: Type Conversions with Complex Datatypes. The presenters is Yukti Kalra.* [Video description ends]

Here we have two variables, the old_num variable with the string 500 and the new_num variable with string 10.

[Video description begins] The TypeConversions-02 page opens in the jupyter notebook. A cell contains the code, code starts: old_num = "500" new_num = "10" remaining_num = old_num - new_num. Code ends. [Video description ends]

We want to subtract the new_num variable from the old_num variable, and store the result in the remaining_num variable.

[Video description begins] *She executes the aforementioned code. The following error message is displayed: TypeError: unsupported operand type(s) for -: 'str' and 'str'.* [Video description ends]

As expected, we'll get an error.

[Video description begins] *In a cell, she enters the code, type(old_num). She executes it. The output reads: str.* [Video description ends]

The type of the old_num variable is a string.

[Video description begins] *In a cell, she enters the code, int(old_num). She executes it. The output reads: 500.* [Video description ends]

We'll convert the old num into an integer using the built-in function int, verify the type of int(old_num).

[Video description begins] *In a cell, she enters the code, type(int(old_num)). She executes it. The output reads: int.* [Video description ends]

Now let's debug the error above. We'll now perform the subtraction operation on the old_num and the new_num, only after converting them into integers.

[Video description begins] *In a cell, she enters the code, code starts: old_num = "500" new_num = "10" remaining_num = int(old_num) - int(new_num) remaining_num. Code ends.* [Video description ends]

Print the remaining_num out to screen, and we get 490, 500 minus 10.

[Video description begins] *In a cell, she enters the code, code starts: old_code = "40.6" new_code = "60.8" total_code = old_code + new_code total_code. Code ends.* [Video description ends]

We have the old_code variable set to the string 40.6, and the new_code variable set to the string 60.8. We'll perform an addition operation on the old_code and the new_code, and store it in the variable total_code. Print total_code to screen.

[Video description begins] *She executes the aforementioned code. The output reads: '40.660.8'. [Video description ends]*

Since we were adding two strings, they have been concatenated.

[Video description begins] In a cell, she enters the code, code starts: old_code = "40.6" new_code = "60.4" total_code = float(old_code) + float(new_code) total_code. Code ends. [Video description ends]

We can convert strings into floating point values using the built-in function, float. Now let's add the float of old_code to the float of new_code. The value stored in the total_code variable is 101.0.

[Video description begins] *She executes the aforementioned code. The output reads: 101.0.* [Video description ends]

It's a floating point.

[Video description begins] *In a cell, she enters the code, code starts: my_float = "4.6" my_int = "6" total = float(my_float) + int(my_int) total. Code ends.* [Video description ends]

Let's initialize two variables, my_float with the string 4.6 and my_int with the string 6. In order to perform addition operation on the variable my_float and the variable my_int, we'll convert the my_float variable into a float and my_int variable into a int. We'll store the result in the variable total.

[Video description begins] *She executes the aforementioned code. The output reads: 10.6.* [Video description ends]

The total is 10.6. Let's try conversions with complex data types.

[Video description begins] *In a cell, she enters the code, code starts: value_str = "python world" value_list = list(value_str) value_list. Code ends.* [Video description ends]

We have a variable, value string, which holds the string python world. Invoke the list function on the value_str variable and store it in the value_list variable. The list function takes in a string and converts it into a list.

[Video description begins] *She executes the aforementioned code. The output reads: ['p', 'y', 't', 'h', 'o', 'n', ' ', 'w', 'o', 'r', 'l', 'd'].* [Video description ends]

The results of this are a little surprising. Individual characters of a string are extracted and they become fields of the list. So the list contains, p, y, t, h, o, n, space, w, o, r, l, d, all as separate characters. Verify the types of the variables value_str and value_list.

[Video description begins] *In a cell, she enters the code, code starts: print('Type of value_str is:', type(value_str)) print('Type of value_list is:', type(value_list)). Code ends. She executes it. The output reads: Type of value_str is: <class 'str'> Type of value_list is: <class 'list'>.* [Video description ends]

Value_str is of the type string. And value_list is of the type list. If we try to do similar stuff on numbers, if we invoke the list function on a series of number, (1, 2, 3 ,4, 5), we'll get an error.

[Video description begins] *In a cell, she enters the code, code starts: list(1, 2, 3, 4, 5). Code ends.* [Video description ends]

The error clearly states, list expected at most 1 arguments, got 5. The list constructor takes a single argument. You can pass iterable objects to the list constructor, such as strings, dictionaries, tuples, sets, and so on.

[Video description begins] *In a cell, she enters the code, code starts: list((1, 2, 3, 4, 5)). Code ends. She executes it. The output reads: [1, 2, 3, 4, 5].* [Video description ends]

Now if you invoke the built-in list function on a tuple of numbers, (1, 2, 3, 4, 5) the tuple will be converted into a list of numbers.

[Video description begins] *In a cell, she enters the code, code starts: my_tuple = ('Apple', 'Orange', 'Mango', 'Banana') my_tuple. Code ends. She executes it. The output reads: ('Apple', 'Orange', 'Mango', 'Banana').* [Video description ends]

We have a tuple, my_tuple, with the elements Apple, Orange, Mango, and Banana. Verify the type of the variable my_tuple. It is indeed a tuple.

[Video description begins] *In a cell, she enters the code, type(my_tuple). She executes it. The output reads: tuple.* [Video description ends]

Let's convert this tuple into a list.

[Video description begins] *In a cell, she enters the code, code starts: my_list = list(my_tuple) my_list. Code ends.* [Video description ends]

Invoke the built-in function list on the variable my_tuple, and store it in the my_list variable. Execute this cell, and we get a list with the elements Apple, Orange, Mango, and Banana.

[Video description begins] *In a cell, she enters the code, type (my_list). She executes it. The output reads: list.* [Video description ends]

Let's check the type of the my_list variable, it is a list.

[Video description begins] *In a cell, she enters the code, code starts: value_str = "python world" value_tup = tuple(value_str) value_tup. Code ends.* [Video description ends]

Similar to the built-in function list, we have a tuple function. If you invoke it on a string, it will convert the string into a tuple of characters. We have the value_str variable initialized with the string python world.

[Video description begins] *She executes the aforementioned code. The output reads: ['p', 'y', 't', 'h', 'o', 'n', ' ', 'w', 'o', 'r', 'l', 'd'].* [Video description ends]

Invoke the tuple function on the value_str variable, and we get a tuple of characters.

[Video description begins] *In a cell, she enters the code, code starts: print('Type of value_str is:', type(value_str)) print('Type of value_tup is:', type(value_tup)). Code ends.* [Video description ends]

Let's check types of variables, value_str and value_tup. The value_str is of type string.

[Video description begins] *She executes the aforementioned code. The output reads: Type of value_str is: <class 'str'>Type of value_tup is: <class 'tuple'>.* [Video description ends]

And the value_tup is of type tuple. Let's see what will happen if you try to pass an integer to the tuple function.

[Video description begins] *In a cell, she enters the code, tuple(5000).* [Video description ends]

This will throw a TypeError. This function accepts a single parameter which is iterable. The error clearly states that the 'int' object is not iterable.

[Video description begins] *In a cell, she enters the code, code starts: my_list = ['Apple', 'Orange', 'Mango', 'Banana'] print(my_list). Code ends. She executes it. The output reads: ['Apple', 'Orange', 'Mango', 'Banana'].* [Video description ends]

Let's initialize a list, my_list, with elements Apple, Orange, Mango, and Banana.

[Video description begins] *In a cell, she enters the code, type(my_list). She executes it. The output reads: list*. [Video description ends]

Verify the type of the variable my_list. It is a list indeed. You can also convert a list to a tuple. Pass in the my_list variable to the built-in function tuple as an input argument.

[Video description begins] *In a cell, she enters the code, code starts: my_tuple = tuple(my_list) print(my_tuple). Code ends*. [Video description ends]

This list simply becomes a tuple.

[Video description begins] *She executes the aforementioned code. The output reads: ('Apple', 'Orange', 'Mango', 'Banana')*. [Video description ends]

Let's verify the type of my_tuple.

[Video description begins] *In a cell, she enters the code, type (my_tuple). She executes it. The output reads: tuple*. [Video description ends]

It is indeed a tuple. Let's see what will happen when we invoke the tuple function on nested list.

[Video description begins] *In a cell, she enters the code, code starts: age_list = [['William', 50], ['Henry', 60], ['James', 90]] age_tuple = tuple(age_list) age_tuple. Code ends*. [Video description ends]

Here we have a nested list stored in the variable age_list. The outer list includes three inner lists. Inner lists have two elements each, the name of the person and the age of the person. We'll invoke the tuple function on the age_list variable and store it in the age_tuple variable. Execute this cell.

[Video description begins] *The output reads: (['William', 50], ['Henry', 60], ['James', 90])*. [Video description ends]

The outer list has become a tuple now, but inner list remain intact.

[Video description begins] *In a cell, she enters the code, code starts: age_tuple = tuple(age_list[0]) age_tuple. Code ends*. [Video description ends]

If you want to convert inner list into tuples, you can use list indexing. Let's convert the very first inner list of the age_list into a tuple.

[Video description begins] *She executes the aforementioned code. The output reads: ('William', 50)*. [Video description ends]

And as you would expect, the list with the elements William and 50, is now a tuple. The type of the age_list is a list.

[Video description begins] *In a cell, she enters the code, code starts: print('Type of age_list is:', type(age_list)) print('Type of age_tuple is:', type(age_tuple)). Code ends. She executes it. The output reads: Type of age_list is: <class 'list'> Type of age_tuple is: <class 'tuple'>*. [Video description ends]

The type of the age_tuple is a tuple. Let's work with the list of tuples.

[Video description begins] *In a cell, she enters the code, code starts: pet_list = [('Dog', 1), ('Cat', 2), ('Cow', 3), ('Goat', 4)]. Code ends*. [Video description ends]

We have a pet_list here. The elements of the outer list are tuples.

[Video description begins] *In a cell, she enters the code, code starts: pet_tuple = tuple(pet_list) pet_tuple. Code ends.* [Video description ends]

Let's pass this pet_list as an input argument to the built-in function tuple, and store it in the variable pet_tuple.

[Video description begins] *She executes the aforementioned code. The output reads: (('Dog', 1), ('Cat', 2), ('Cow', 3), ('Goat', 4)).* [Video description ends]

Print out the pet_tuple. Now we have a tuple of tuples.

[Video description begins] *In a cell, she enters the code, code starts: age_tuple = ('Leo', 18, 'Aaron', 25, 'Easton', 34, 'Jordan', 30). Code ends. [Video description ends]*

The outer list has been converted into a tuple. Let's initialize an age_tuple.

[Video description begins] In a cell, she enters the code, age_tuple. She executes it. The output reads: ('Leo', 18, 'Aaron', 25, 'Easton', 34, 'Jordan', 30). [Video description ends]

The tuple contains ages of Leo, Aaron, Easton, and Jordan. Let's convert this tuple into a dictionary by calling the built-in function dict.

[Video description begins] *In a cell, she enters the code, dict(age_tuple). She executes it.* [Video description ends]

We get an error, the ValueError.

[Video description begins] *In a cell, she enters the code, code starts: age_tuple1 = (('Leo', 18), ('Aaron', 25), ('Easton', 34), ('Jordan', 30)) age_tuple1. Code ends.* [Video description ends]

Let's fix this error. Now we have a tuple of tuples stored in the variable age_tuple1. Notice here we have four inner tuples with two fields each. Leo is 18 years old. Aaron is 25 years old. Easton is 34 years old.

[Video description begins] *She executes the aforementioned code. The output reads: (('Leo', 18), ('Aaron', 25), ('Easton', 34), ('Jordan', 30)).* [Video description ends]

And Jordan is 30 years old. Now pass this age_tuple1 into the built-in dict function. This returns a dictionary.

[Video description begins] *In a cell, she enters the code, dict(age_tuple1).* [Video description ends]

Leo, Aaron, Easton, and Jordan are keys in the dictionary.

[Video description begins] *She executes the aforementioned code. The output reads: {'Leo': 18, 'Aaron': 25, 'Easton': 34, 'Jordan': 30}.* [Video description ends]

18, 25, 34, 30, are values in the dictionary. Now we know that a tuple of tuples can be converted into a dictionary.

[Video description begins] *In a cell, she enters the code, code starts: age_tuple2 = (('Leo', 18), ('Aaron', 25), ('Easton', 34), (['Jordan', 'John'], 30)) age_tuple2. Code ends.* [Video description ends]

We have an another tuple of tuples, the age_tuple2. Notice the fourth inner tuple. The fields of this tuple are a list of names and the age 30.

[Video description begins] *She highlights the code, (['Jordan', 'John'], 30)).* [Video description ends]

Both Jordan and John are 30 years old.

[Video description begins] *She executes the code, code starts: age_tuple2 = (('Leo', 18), ('Aaron', 25), ('Easton', 34), (['Jordan', 'John'], 30)) age_tuple2. Code ends. The output reads: (('Leo', 18), ('Aaron', 25), ('Easton', 34), (['Jordan', 'John'], 30))*. [Video description ends]

Let's see if we can convert this age_tuple2 into a dictionary. Invoke the built-in function dict on the age_tuple2. We get an error.

[Video description begins] *In a cell, she enters the code, dict(age_tuple2)*. [Video description ends]

The error clearly states, unhashable type: 'list'. We cannot have a list as a key in a dictionary. Let's take a look at another tuple, the age_tuple3. Notice the fourth inner tuple.

[Video description begins] *In a cell, she enters the code, code starts: age_tuple3 = (('Leo', 18), ('Aaron', 25), ('Easton', 34), ('Jordan', ['John', 30])) age_tuple3. Code ends*. [Video description ends]

The second field in the fourth inner tuple is a list, with the elements John and 30.

[Video description begins] *She highlights the code, ('Jordan', ['John', 30])). She executes the code, code starts: age_tuple3 = (('Leo', 18), ('Aaron', 25), ('Easton', 34), ('Jordan', ['John', 30])) age_tuple3. Code ends. The output reads: (('Leo', 18), ('Aaron', 25), ('Easton', 34), ('Jordan', ['John', 30]))*. [Video description ends]

Pass the age_tuple3 variable into the built-in function dict.

[Video description begins] *In a cell, she enters the code, dict(age_tuple3)*. [Video description ends]

This time, we won't get any errors, because we can have a list as a value in the dictionary.

[Video description begins] *She executes the aforementioned code. The output reads {'Leo': 18, 'Aaron': 25, 'Easton': 34, 'Jordan': ['John', 30]}*. [Video description ends]

You can also convert a list of lists into a dictionary.

[Video description begins] *In a cell, she enters the code, code starts: age_list = [['William', 50], ['Joanne', 60], ['Maria', 90]] age_dict = dict(age_list) age_dict. Code ends*. [Video description ends]

Here we have a nested list stored in the variable age_list.

[Video description begins] *She highlights the line of code, code starts: age_list = [['William', 50], ['Joanne', 60], ['Maria', 90]]. Code ends*. [Video description ends]

Invoke the dict function on the age_list. The age_dict returns a dictionary. William, Joanne, Maria are keys in the dictionary.

[Video description begins] *She executes the code, code starts: age_list = [['William', 50], ['Joanne', 60], ['Maria', 90]] age_dict = dict(age_list) age_dict. Code ends. The output reads: {'William': 50, 'Joanne': 60, 'Maria': 90}*. [Video description ends]

50, 60, 90 are the corresponding values.

[Video description begins] *In a cell, she enters the code, code starts: print('Type of age_list is:', type(age_list)) print('Type of age_dict is:', type(age_dict)). Code ends. She executes it. The output reads: Type of age_list is: <class 'list'> Type of age_dict is: <class 'dict'>*. [Video description ends]

The type of the variable age_ list is a list. And type of the age_dict variable is a dictionary.

Type Conversions and Base Conversions

[Video description begins] *Topic title: Type Conversions and Base Conversions. The presenter is Yukti Kalra. [Video description ends]*

Python also supports the set data structure.

[Video description begins] The TypeConversions-02 page opens in the jupyter notebook. [Video description ends]

Sets basically remove all the duplicates from any list.

[Video description begins] *A cell contains the code, set('apple').* [Video description ends]

A set contains no duplicates. So the set of the string apple will contain all the characters with an apple, but the duplicates have been removed.

[Video description begins] *She executes the aforementioned code. The output reads: {'a', 'e', 'l', 'p'}.* [Video description ends]

The set will not contain the extra p. Let's say we have a tuple of fruits with Apple, Orange, Mango, and Banana.

[Video description begins] *In a cell, she enters the code, code starts: fruit_tuple = ('Apple', 'Orange', 'Mango', 'Banana', 'Banana') fruit_set = set(fruit_tuple) fruit_set. Code ends.* [Video description ends]

Notice the tuple contains two bananas. Let's create a set of fruits by passing the fruit_tuple into the built-in function set, and you can see the fruit_set has only four elements.

[Video description begins] *She executes the aforementioned code. The output reads: {'Apple', 'Banana', 'Mango', 'Orange'}.* [Video description ends]

The extra banana has been removed.

[Video description begins] *In a cell, she enters the code, type(fruit_set). She executes it. The output reads: set. [Video description ends]*

Check the type of the variable fruit_set, it is indeed a set.

[Video description begins] In a cell, she enters the code, hex(50). [Video description ends]

There are ways to convert into just represented as these ten format into other formats as well.

[Video description begins] *She executes the code, hex(50). The output reads: '0x32'. [Video description ends]*

Let's find the hexadecimal value for the integer 50 in base 10 '0x32' is the hexadecimal value.

[Video description begins] In a cell, she enters the code, hex(60). She executes it. The output reads: '0x3c'. [Video description ends]

Let's check the hexadecimal value for the integer 60. The hexadecimal value is '0x3c'.

[Video description begins] *In a cell, she enters the code, oct(50).* [Video description ends]

The built-in function oct takes in an integer and returns its octal representation in a string format.

[Video description begins] *She executes the aforementioned code. The output reads: '0o62'.* [Video description ends]

The octal representation of the number 50 is '0o62'.

[Video description begins] *In a cell, she enters the code, oct(9). She executes it. The output reads: '0o11'.[Video description ends]*

Let's see the octal representation of the number 9. It is '0o11'.

[Video description begins] In a cell, she enters the code, bin(5). [Video description ends]

We can also convert an integer into a binary format by using the built-in function bin.

[Video description begins] *She executes the aforementioned code. The output reads: '0b101'.* [Video description ends]

The binary representation of the integer 5 is 0b101.

[Video description begins] *In a cell, she enters the code, bin(10). She executes it. The output reads: '0b1010'. [Video description ends]*

Check the binary representation of the integer 10.

[Video description begins] In a cell, she enters the code, int('0b1010' ,2). [Video description ends]

The built-in function int in Python converts a number in a given base to an integer object. The int method takes in two arguments. The first input argument is the number or the string that has to be converted to an integer object. And the second input argument is the base of the number passed in. Let's use the binary format for the number 10 and pass it in the int function. The base is 2, this returns the integer 10.

[Video description begins] *In a cell, she enters the code, int('0b1010', 0).* [Video description ends]

Once more, let's pass the binary format for the integer 10 into the int function. Notice the base here, the base is 0. The base 0 means to interpret exactly as a code literal, so this will also return the integer 10.

[Video description begins] *In a cell, she enters the code, ('0x101', 16).* [Video description ends]

Let's find out the integer value for this hexadecimal literal. The base is 16, this returns the integer 257.

[Video description begins] *In a cell, she enters the code, ord('a').* [Video description ends]

The built-in function ord returns an integer representing the Unicode point for the given Unicode character.

[Video description begins] *She executes the aforementioned code. The output reads: 97.* [Video description ends]

The Unicode point for the lower case character a is 97.

[Video description begins] *In a cell, she enters the code, ord('A'). She executes it. The output reads: 65.[Video description ends]*

The Unicode character for the upper case A is 65.

[Video description begins] In a cell, she enters the code, chr(100). [Video description ends]

The character method returns a string representing a character whose Unicode point is an integer. Let's check the character corresponding to the Unicode point 100.

[Video description begins] *She executes the aforementioned command. The output reads:* '*d*'. [Video description ends]

It is the lowercase d.

[Video description begins] *In a cell, she enters the code, chr(97)*. [Video description ends]

Let's check the character corresponding to the Unicode point 97.

[Video description begins] *She executes the aforementioned command. The output reads:* '*a*'. [Video description ends]

It is the lowercase a.

[Video description begins] *In a cell, she enters the code, complex(1j)*. [Video description ends]

As suggested by the name, the complex method returns a complex number. This method takes two optional parameters and returns a complex number. The first parameter is called a real and second as imaginary.

[Video description begins] *She executes the aforementioned command. The output reads:* '*1j*'. [Video description ends]

The complex number for 1j is 1j.

[Video description begins] *In a cell, she enters the code, complex(10). She executes it. The output reads: (10+0j)*. [Video description ends]

The complex representation of the integer 10 is 10+0j.

[Video description begins] *In a cell, she enters the code, complex(1,2). She executes it. The output reads: (1+2j)*. [Video description ends]

The complex representation of the numbers 1, 2 is 1+2j.

Basic Programs - Part 1

[Video description begins] *Topic title: Basic Programs - Part 1. The presenter is Yukti Kalra.[Video description ends]*

We've learned a bunch of concepts so far, it's time to apply them in practice.

[Video description begins] The SomeBasicPrograms page opens in the jupyter notebook. [Video description ends]

Here are some basic programs that will help you sharpen your programming logic. Let's write code to find out the total savings in a month for an employee.

[Video description begins] *In a cell, she enters the code, code starts: salary = int(input("Enter salary amount: ")) expenses = int(input("Enter expenses: ")) savings = salary - expenses print("My total savings in a month is: ", savings). Code ends*. [Video description ends]

We'll accept inputs from the user for the salary variable and the expenses variable. The input function returns strings, we need to convert them into an integer, invoke the int function on the input function. Whenever you accept an input from the user it's a good practice to write out a friendly message which prompts the user for the input he has to provide. We'll calculate the savings by using a simple subtraction operation, savings = salary -

expenses, then we'll print the savings out to screen. There's no error checking that we've added to this code. We haven't studied it yet. We haven't learnt how we can check whether the number is an integer or not and how we can handle situations where the input is not a number.

[Video description begins] *She enters the salary amount: 700. The output reads: Enter salary amount: 700 Enter expenses: 230 My total savings in a month is: 470.* [Video description ends]

Execute this bit of code, the salary amount is $700, the expenses are $230, and the savings are 700 - 230, that is $470. Let's move on to the next program.

[Video description begins] *In a cell, she enters the code, code starts: first_num = int(input("Enter first variable: ")) second_num = int(input("Enter second variable: ")) first_num = first_num + second_num second_num = first_num - second_num first_num = first_num - second_num print ("\nFirst_num is:", first_num , "\nSecond_num is:", second_num). Code ends.* [Video description ends]

Let's try to program to exchange values of two numbers without using a temporary variable. The first_num and the second_num will be user inputs. Execute this cell, the first_ num is 20 and the second_num is 25.

[Video description begins] *The output reads: Enter first variable: 20 Enter second variable: 25 First_num is: 25 Second_num is: 20.* [Video description ends]

We'll add the second_num to the first_num and store it in the first_num variable. Now, the first_num variable holds the value 45. We'll subtract the second_num from the first_num and store it in the second_num variable. 45 - 25 is 20. Now the second_num is 20. Now we will subtract the second_num from the first_num, that is 45 - 20, and we store it in the first_num variable. Now the first_num is 25.

[Video description begins] *In the output, she highlights First_num is: 25 Second_num is: 20.* [Video description ends]

You can see the numbers exchanged now. The first_num is 25, the second_num is 20.

[Video description begins] *In a cell, she enters the code, code starts: num = int(input("Enter a number: ")) temp1 = str(num) temp2 = temp1 + temp1 temp3 = temp1 + temp1+ temp1 total = num + (int(temp2) + int(temp3)) print(temp1, '+', temp2, '+', temp3, '=', total). Code ends.* [Video description ends]

Let's write a program that accepts a number n from the user and it finds the sum of n+nn+nnn. We'll accept an input from the user and store it in the variable num. Convert this num into a string and store it in the variable temp1. Remember, the addition of strings results into concatenation of strings. We'll concatenate the num to form nn and store it in the variable temp2. Similarly, to form nnn, we'll concatenate the temp1 variable thrice, and store it in the variable temp3. In order to find the sum of the number in the form n + nn + nnn, we'll convert temp2 and temp3 into integers and add it to num. We'll store the result in the variable total.

[Video description begins] *She enters the value: 12. The output reads: Enter a number: 12 12 + 1212 + 121212 = 122436.* [Video description ends]

Execute this cell and you can see the total has been calculated and printed out to screen. Let's work with two numbers here.

[Video description begins] *In a cell, she enters the code, code starts: num_1 = int(input("Enter first number: ")) num_2 = int(input("Enter second number: ")) Quotient = int(num_1 / num_2) print("\nQuotient:", Quotient) Remainder = num_1 % num_2 print("Remainder:", Remainder). Code ends.* [Video description ends]

We'll write a program to find the quotient and the remainder when the first number is divided by the second number. We'll accept two numbers as inputs from the user. We'll obtain the quotient using the division operator and store it in the variable quotient. Since we want the integer quotient, we invoke the int function on the division. We'll print the quotient out to screen. We'll find the remainder of this division by using the modulus operator. And then we'll print the remainder out to screen. The first number is 13. The second number is 4.

[Video description begins] *She executes the aforementioned code. The output reads: Enter first number: 13 Enter second number: 4 Quotient: 3 Remainder: 1.* [Video description ends]

The quotient obtained from dividing 13 by 4 is 3 and the remainder is 1. Write a program that will calculate the simple interest.

[Video description begins] *In a cell, she enters the code, code starts: principle = float(input("Enter the principle amount: ")) time = int(input("Enter the time(years): ")) rate = float(input("Enter the rate: ")) simple_int = (principle * time * rate) / 100 print("The simple interest is:", simple_int). Code ends.* [Video description ends]

We'll accept inputs from the user for the principle amount, the time, and the rate. We calculate the simple interest by multiplying the principle amount to time, and rate, and then we divide the result by 100. The principle amount is $800. The time is 6 years, and the rate is 3%.

[Video description begins] *She executes the aforementioned code. The output reads: Enter the principle amount: 800 Enter the time(years): 6 Enter the rate: 3 The simple interest is: 144.0.* [Video description ends]

The simple interest is $144.

[Video description begins] *In a cell, she enters the code, code starts: cm = int(input("Give the height in centimeters: ")) inches = 0.394 * cm feet = 0.0324 * cm print ("The length in inches: ", round(inches, 2)) print ("The length in feet: ", round(feet, 2)). Code ends.* [Video description ends]

Now, let's create a conversion calculator that accepts the height of a user in centimeters and converts it into inches and feet. 1 centimeter is equal to 0.394 inches, so we'll multiply the height of the user by 0.394 to convert it into inches. Similarly, to obtain the height in feet, we'll multiply the height in centimeters to 0.0324.

[Video description begins] *The output reads: Give the height in centimeters: 168 The length in inches: 66.19 The length in feet: 5.44.* [Video description ends]

Execute this cell, the height of the user is 168 centimeters, the height in inches is 66.19 inch, and the height in feet is 5.44.

Basic Programs - Part 2

[Video description begins] *Topic title: Basic Programs - Part 2. The presenter is Yukti Kalra.Video description ends]*

Here we have a list of cars, Toyota Camry, Honda Accord, Honda Civic, and Toyota Corolla.

[Video description begins] The SomeBasicPrograms page opens in the jupyter notebook. A cell contains the code, code starts: cars_list = ["Toyota Camry", "Honda Accord", "Honda Civic", "Toyota Corolla"] print("List of cars before the swap: ", cars_list) cars_list_temp = cars_list[0] cars_list[0] = cars_list[2] cars_list[2] = cars_list_temp print("List of cars after the swap: ", cars_list). Code ends. [Video description ends]

We'll write a program to swap the positions of Toyota Camry and Honda Civic.

[Video description begins] *She highlights the line of code, cars_list_temp = cars_list[0].* [Video description ends]

We'll use list indexing here. We'll store the first element of the list of cars in a temporary variable called the cars_list_temp. Now the cars_list_temp variable contains Toyota Camry.

[Video description begins] *She highlights the line of code, cars_list[0] = cars_list[2].* [Video description ends]

Now we'll store the element at index 2 of the cars list at index 0. Now the element stored at index 0 of the cars_list is Honda Civic. Remember that the cars_list_temp variable has Toyota Camry.

[Video description begins] *She highlights the line of code, cars_list[2] = cars_list_temp.* [Video description ends]

We'll copy over the value stored in the cars_list_temp variable to the element at index 2 of the cars_ list. Execute this cell.

[Video description begins] *The output reads: List of cars before the swap: ['Toyota Camry', 'Honda Accord', 'Honda Civic', 'Toyota Corolla'] List of cars after the swap: ['Honda Civic', 'Honda Accord', 'Toyota Camry', 'Toyota Corolla'].* [Video description ends]

Notice the list of cars after the swap. The Honda Civic is at index 0, and the Toyota Camry is at index 2.

[Video description begins] *In a cell, she enters the code, code starts: cars_list = ["Toyota Camry", "Honda Accord", "Honda Civic", "Toyota Corolla"] print("list of cars before the swap: ", cars_list) car1 = 1 car2 = 2 cars_list[car1], cars_list[car2] = cars_list[car2], cars_list[car1] print("list of cars after the swap: ", cars_list). Code ends.* [Video description ends]

Now, let's try to swap two elements of the list without using a temporary variable. We want to swap the positions of Honda Accord and Honda Civic.

[Video description begins] *She highlights the line of code, code starts: cars_list = ["Toyota Camry", "Honda Accord", "Honda Civic", "Toyota Corolla"]. Code ends.* [Video description ends]

We have two variables here, car1 = 1, car2 = 2. Remember that Python also allows you to assign several values to several variables within the same line.

[Video description begins] *She highlights the line of code, code starts: cars_list[car1], cars_list[car2] = cars_list[car2], cars_list[car1]. Code ends.* [Video description ends]

So we assign the car at index 2 that is Honda Civic to be at index 1 and we assign the car at index 1, that is Honda Accord, to be at index 2 of the cars_list. Execute this cell.

[Video description begins] *The output reads: list of cars before the swap: ['Toyota Camry', 'Honda Accord', 'Honda Civic', 'Toyota Corolla'] list of cars after the swap: ['Toyota Camry', 'Honda Civic', 'Honda Accord', 'Toyota Corolla'].* [Video description ends]

Notice the list of cars after the swap, the cars have been swapped, Honda Civic is at index 1 and Honda Accord is at index 2. Let's write code to check duplicate elements in a list and remove them.

[Video description begins] *In a cell, she enters the code, code starts: list_student = ["Sofia", "Ella", "Samuel", "Ella", "Aiden", "Sofia"] print("Student list: ", list_student) print("Number of Students: ", len(list_student)) student_set = set(list_student) print("\nNew student list: ", list(student_set)) print("Length of modified students list: ", len(student_set)) print("There are %s duplicate elements"%(len(list_student) - len(student_set))). Code ends.* [Video description ends]

If you remember from the previous classes, we used the set to remove duplicates from a list. Here we have a list of students, Sofia, Ella, Samuel, Ella, Aiden, and Sofia. We'll print this list out to screen and we'll also print the length of this list in order to remove duplicates from this list we'll use the set inbuilt function, then we'll print this set out to screen. Notice we will invoke the list function on this student set as we want a list. We'll also print the length of the student set. We'll obtain the number of duplicate elements in the original list by subtracting the length of the student set from the length of the student list. Execute this cell.

[Video description begins] *The output reads: Student list: ['Sofia', 'Ella', 'Samuel', 'Ella', 'Aiden', 'Sofia'] Number of Students: 6 New student list: ['Aiden', 'Samuel', 'Sofia', 'Ella'] Length of modified students list: 4 There are 2 duplicate elements.* [Video description ends]

The number of students in the original list was six. Notice the new list of students, the original order has been lost because we had converted it to a set. The length of the student_set is four. There were two duplicate elements in the original list which were removed using the set function. We can also check the duplicate elements in a list by using the if-else statement. Using an if-else statement makes the code more crisp.

[Video description begins] *In a cell, she enters the code, code starts: list_str = ["Sofia","Ella","Samuel","Ella","Aiden","Sofia"] len_list_str = len(list_str) len_set_str = len(set(list_str)) if len_list_str == len_set_str: print("There are no duplicate elements") else: print("There are {} duplicate elements".format(len_list_str - len_set_str)). Code ends.* [Video description ends]

We have a list of strings here. We have two variables here, len_list_str and len_set_str. We'll convert the list of strings into a set in order to remove the duplicates. Now we'll check for duplicates using the if-else statement if length of the list is equal to the length of the set. If this evaluates to true, then There are no duplicate elements. If this condition evaluates to false, we'll move into the else block. In the else block, we'll find out how many duplicate elements are there in the list, by subtracting the len_list_str from the len_set_str. The If condition evaluates to false and there are two duplicate elements as printed out to screen.

[Video description begins] *In a cell, she enters the code, code starts: num_value = 50 print("The number stored in num_value is :", num_value) if not num_value % 2 == 0: print("The number is an odd number") else: print("The number is an even number"). Code ends.* [Video description ends]

Let's check if a number is an even or an odd number, using the if-else statement. The variable num_value is set to 50. The if condition evaluates if the num_value is not divisible by 2, then the number is an odd number, else the number is an even number. When 50 is divided by 2 it gives out the remainder 0, therefore, the if condition evaluates to false, The number is an even number is printed out to the screen.

[Video description begins] *She executes the aforementioned code. The output reads: The number stored in num_value is : 50 The number is an even number.* [Video description ends]

Now let's try to program that will accept comma separated sequence of numbers from the user, and then it will generate a list and a tuple which will contain every number.

[Video description begins] *In a cell, she enters the code, code starts: values = input("Enter some comma separated numbers : ") list_value = values.split(",") tuple_value = tuple(list_value) print("List : ", list_value) print("Tuple : ", tuple_value). Code ends.* [Video description ends]

The input function will return strings which we will store in the variable values. We'll use the .split method, it returns a list of strings after splitting the given strings by the specified separator. We'll store this list of strings in the list_value variable. We'll convert the list into a tuple by invoking the tuple function on the list_value variable and storing it in the tuple_value variable. Let's enter some comma separated numbers 12, 23, 45, 56. This is a string. We'll separate these numbers on the comma and we'll get a list of strings of these numbers by calling the .split method on the values variable.

[Video description begins] *The output reads: Enter some comma separated numbers : 12, 23, 45, 56 List : ['12', '23', '45', '56'] Tuple : ('12', '23', '45', '56').* [Video description ends]

And we also get a tuple by converting this list into a tuple. Now let's try to program that will accept a sequence of comma separated words from the user and then sort the words alphabetically in a list.

[Video description begins] *In a cell, she enters the code, code starts: list_words = input("Enter some comma separated words : ") words = list_words.split(",") new_list_words = sorted(words) print (new_list_words). Code* *ends*. [Video description ends]

We'll split the words on the comma separator and store it in the variable words. Execute this cell. The words are Kiwi, Apple, Banana. We'll sort the words by using the built in function sorted, and storing it in the new_list_words variable.

[Video description begins] *The output reads: Enter some comma separated words: Kiwi, Apple, Banana ['Apple', 'Banana', 'Kiwi']*. [Video description ends]

And as you can see when we print, the new_list_words out to screen, Apple, Banana, Kiwi are alphabetically sorted. Now let's say you want to write a program, that will remove all the duplicate words in a sentence and also sort the words alphabetically. Generally, words and sentences are white space separated.

[Video description begins] *In a cell, she enters the code, code starts: sentence = input("Enter some whitespace-separated words : ") words = sentence.split(" ") set_of_words = set(words) sorted_set_of_words = sorted(set_of words) print(" ".join(sorted_set_of_words)). Code ends*. [Video description ends]

We'll accept a sentence from the user with white space separated words. We'll store the input from the user in the variable sentence. We'll split the sentence on the white space which will give us the words in the sentence. We'll store this list of words in the variable words. In order to remove the duplicates in the sentence, we'll invoke the set function on the words variable and store it in the set_of_words variable. Then we'll sort the set of words alphabetically, by invoking the inbuilt sorted function and we'll store it in the variable sorted_set_of_words. Then we'll use the .join method on the white space in order to form a sentence out of a sorted_set_of_words. Execute this cell, enter a sentence, what lies behind us and what lies before us are tiny matters compared to what lies within us.

[Video description begins] *The output reads: Enter some whitespace-separated words : what lies behind us and what lies before us are tiny matters compared to what lies within us and are before behind compared lies matters tiny to us what within*. [Video description ends]

And as you can see on screen, the duplicates have been removed and the words have been sorted alphabetically.

Basic Programs - Part 3

[Video description begins] *Topic title: Basic Programs - Part 3. The presenter is Yukti Kalra. [Video description ends]*

Let's write a program to create a dictionary such that the keys of the dictionary are numbers 1 to 3, and the values are square of the keys.

[Video description begins] The SomeBasicPrograms page opens in the jupyter notebook. [Video description ends]

Initialize an empty dictionary by invoking the dict function and store it in the variable num_dict.

[Video description begins] *In a cell, she enters the code, code starts: num_dict = dict() num_dict[1] = 1 num_dict[2] = 2 ** 2 num_dict[3] = 3 ** 2 print (num_dict). Code ends*. [Video description ends]

Now we'll add key value pairs to this dictionary using the assignment operator. Here the keys are 1, 2, and 3. We'll use the assignment operator to assign values to these keys. The values are squares of their corresponding keys. 1 square is 1, so we'll not perform the square operation on 1. Print out the num_dict variable.

[Video description begins] *She executes the aforementioned code. The output reads: {1: 1, 2: 4, 3: 9}*. [Video description ends]

And you can see this dictionary now contains keys 1, 2, and 3, and values 1, 4, and 9.

[Video description begins] *In a cell, she enters the code, code starts: string_1 = input("Enter first string: ") string_2 = input("Enter second string: ") set_1 = set(string_1) set_2 = set(string_2) common_char = set_1.intersection(set_2) print("\nCommon letters: ", list(common_char)). Code ends.* [Video description ends]

Now let's accept two strings as inputs from the user. We'll write code to check common characters between two strings. Execute this cell. The string_1 is hello and the string_2 is python.

[Video description begins] *The output reads: Enter first string: hello Enter second string: python Common letters: ['h', 'o'].* [Video description ends]

First, we'll remove the duplicates, if any. So we'll invoke the set function on string 1 and 2, store it in variables set_1 and set_2 respectively. We'll use the intersection function here to get the common characters. The intersection method returns a new set with elements that are common to all sets. We'll store this intersection in the variable Common_char. Print Common characters out to screen, h and o common letters present in the strings hello and python. Let's write a program to divide a given list into two halves.

[Video description begins] *In a cell, she enters the code, code starts: list_num = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21] len_of_list = len(list_num) print("Numbers before slicing: %s" %list_num) print("Length of numbers before slicing: ", len_of_list) half = int(len_of_list / 2) list_num1 = list_num[:half] list_num2 = list_num[half:] print("\nFirst half: %s" %list_num1) print("Second half: %s" %list_num2). Code ends.* [Video description ends]

We have a list here stored in the variable list_num, we'll find the length of this list by invoking the len function.

[Video description begins] *She executes the aforementioned code. The output reads: Numbers before slicing: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21] Length of numbers before slicing: 12 First half: [10, 11, 12, 13, 14, 15] Second half: [16, 17, 18, 19, 20, 21].* [Video description ends]

There are 12 elements in this list. Initialize a variable half that will store the value of len_of_list divided by 2. We'll use the list slicing operation here to divide the list into two halves. We want the list_num1 variable to contain the First half of the elements of the list_num. The variable half contains the value 6. We'll store the elements from index 0 to 5 in the list_num1 variable, as the sixth element will be non-inclusive. In the list slicing operation, and the Second half of the list_num will be stored in the list_num2 variable, starting from the element at index 6 till the end of the list. Let's accept a number as an input from the user.

[Video description begins] *In a cell, she enters the code, code starts: num = float(input("Enter a number: ")) if num >= 0: if num == 0: print("Zero") else: print("Positive number") else: print("Negative number"). Code ends.* [Video description ends]

And we'll write code to check whether the number is a positive number, a negative number, or a zero. We'll use nested if-else statements here, if the number is more than or equal to 0. If this condition evaluates to true, we'll move into the if block, else we will print it's a Negative number. If the first if statement evaluates to true, we'll evaluate the nested if-else statements. If number is equal to 0, we'll print Zero to screen. If this condition evaluates to false, we'll move into the nested else block and Positive number will be printed out to screen. Let's check for the number 23.

[Video description begins] *She executes the aforementioned code. The output reads: Enter a number: 23 Positive number.* [Video description ends]

The first if block evaluates to true, the number is greater than 0, then we'll check for the nested if-else conditions. The nested if condition evaluates to false, the number is not equal to 0. We'll move into the else block, Positive number is printed out to screen. Let's check for the number -12.

[Video description begins] *She executes the aforementioned code again. The output reads: Enter a number: -12 Negative number.* [Video description ends]

The first if condition evaluates if the number is greater than or equal to 0. This evaluates to false. We directly move into the outer else. Negative number is printed out to screen.

[Video description begins] *In a cell, she enters the code, code starts: var = 3 if (type(var) == int): print("Type of the variable is Integer") elif (type(var) == float): print("Type of the variable is Float") elif (type(var) == complex): print("Type of the variable is Complex") else: print("Type of the variable is Unknown"). Code ends.* [Video description ends]

Let's write a program to check the type of a given number. We have the variable var = 3. We'll use if-else statements to check the type of the variable var. Execute this cell.

[Video description begins] *The output reads: Type of the variable is Integer.* [Video description ends]

The first if condition evaluates if the type of the var is equal to int. This evaluates to true. Type of the variable is Integer is printed out to screen. Let's check for another value. Now the var is equal to 1+2j.

[Video description begins] *She alters the line of code from var = 3 to var = 1+2j.* [Video description ends]

The first if condition will check if the type of the var is equal to int. This evaluates to false. We'll move into the next elif block. The elif condition evaluates if the type of the var is equal to float. This evaluates to false. We'll move into the next elif block. Now this elif condition evaluates if the type of the var is equal to complex. This evaluates to true. Type of the variable is complex is printed out to screen. Let's accept a number as an input from the user. Now we'll check if the number is even or odd.

[Video description begins] *In a cell, she enters the code, code starts: num = int(input("Enter a number: ")) if (num % 2) == 0: print("%d is Even" % num) else: print("%d is Odd" % num). Code ends.* [Video description ends]

We'll store the user input in the variable num. Remember to invoke the int function on the input. The if condition evaluates if the number is divided by 2 and it gives out a remainder 0, it's an Even number. Otherwise, it's an Odd number. Let's check for the number 42.

[Video description begins] *She executes the aforementioned code. The output reads: Enter a number: 42 42 is Even.* [Video description ends]

It is an Even number. Let us also check for the number 59. 59 is Odd.

[Video description begins] *She executes the aforementioned code again. The output reads: Enter a number: 59 59 is Odd.* [Video description ends]

Let's write code to make a calculator.

[Video description begins] *In a cell, she enters the code, code starts: num1 = float(input("Enter the first number: ")) operator = input("Operator: ") num2 = float(input("Enter the second number: ")) if operator == "+": print("Addition: ", num1 + num2) elif operator == "-": print("Subtraction: ", num1 - num2) elif operator == "*": print("Multiplication: ", num1 * num2) elif operator == "/": print("Division: ", num1 / num2) else: print("This is not a valid operator"). Code ends.* [Video description ends]

We'll accept two numbers as inputs from the user and store it in variables num1 and num2 respectively. We'll also accept an operator from the user to perform the desired operation. Let's see how this calculator will work. If the operator is equal to the symbol plus, the num2 will be added to num1. If this condition evaluates to false, we'll move into the elif block to check if the operator is equal to the subtraction symbol. If this evaluates to true, num2 will be subtracted from num1. If this also evaluates to false, we'll move into the next elif block, which will check if the operator is equal to the symbol star. If this evaluates to true, then num1 will be multiplied to num2. If this evaluates to false, we'll move into the next elif block, which will check if the operator is equal to the forward slash. If this evaluates to true, then num1 will be divided by num2. If none of the above conditions are met, then This is not a valid operator will be printed out to screen. The first number is 27. The operator is equal to the symbol star, and the second number is 12.

[Video description begins] *She executes the aforementioned code. The output reads: Enter the first number: 27 Operator: * Enter the second number: 12 Multiplication: 324.0.* [Video description ends]

The second elif block evaluates to true. The calculator performs the multiplication operation. 27 multiply by 12 is 324. Let's perform one more operation. The first number is 12, the operator is a forward slash and the second number is 4.

[Video description begins] *She executes the aforementioned code again. The output reads: Enter the first number: 12 Operator: / Enter the second number: 4 Division: 3.0.* [Video description ends]

The third elif block evaluates to true. 12 divided by 4 is 3.

[Video description begins] *In a cell, she enters the code, code starts: num_1 = int(input("Enter a number: ")) if num_1 % 5 == 0 and num_1 % 7 == 0: print("This number is divisible by both 5 and 7") elif num_1 % 5 == 0: print("This number is divisible by 5") elif num_1 % 7 == 0: print("This number is divisible by 7") else: print("This number is neither divisible by 5 nor 7"). Code ends.* [Video description ends]

Let's write a program that will check the divisibility of a given number by 5, 7, or both. We'll store the input from the user in the variable num_1. The first if condition checks if the number is divisible by both 5 and 7. We have to check the divisibility by 5 and 7 first so we know if a number is divisible by both. If num_1 divided by 5 gives out the remainder 0 and if num_1 divided by 7 gives out the remainder 0, then we'll print this number is divisible by both 5 and 7. If this evaluates to false, we'll move into the next elif block to check if the num_1 is divisible by 5. If this evaluates to true, This number is divisible by 5 will be printed out to screen. If this evaluates to false, we'll move into the next elif block to check if this number is divisible by 7. If this also evaluates to false, we'll move into the else block, This number is neither divisible by 5 nor 7. Let's check for the number 40.

[Video description begins] *The output reads: Enter a number: 40 This number is divisible by 5.* [Video description ends]

The first elif block evaluates to true, This number is divisible by 5. Let's check the divisibility of another number by 5 and 7. The number entered is 35.

[Video description begins] *The output reads: Enter a number: 35 This number is divisible by both 5 and 7.* [Video description ends]

This number is divisible by both 5 and 7. The first if statement evaluates to true.

Basic Programs - Part 4

[Video description begins] *Topic title: Basic Programs - Part 4. The presenter is Yukti Kalra.[Video description ends]*

Let's write a program to calculate a dog's age in the dog years.

[Video description begins] The SomeBasicPrograms page opens in the jupyter notebook. [Video description ends]

The logic behind this code is for the first two years, a dog year is equal to 10.5 human years.

[Video description begins] *A cell contains the code, code starts: human_age = int(input("Enter dog's age in human years: ")) if human_age < 0: print("Age must be a positive number.") exit() if human_age <= 2: dog_age = human_age * 10.5 else: dog_age = 21 + (human_age - 2)*4 print("The dog's age in dog years is", dog_age). Code ends.* [Video description ends]

After that, each dog year equals four human years. There are other ways to calculate the dog years, this is one of the metrics. First we'll accept an input from the user, that is a dog's age in human years. We'll store this input in the human_age variable. First of all we want the age to be a positive number, so we'll check if the human_age is less than 0. If this evaluates to true, age must be a positive number, it will be printed out to screen, and exit method will stop the code execution. You cannot proceed further if this statement evaluates to true. If this statement evaluates to false, we'll move into the next if block, which will check if the human_age is less than or equal to 2 years. Then, we'll calculate the dog's age by multiplying the human_age by 10.5. If the dog is more than 2 years old, then we'll move into the else block. The dog's age will be calculated in the following manner. For the first two years, the dog's age is 21, and for the remaining years, the age will be multiplied by 4. Let's check the dog's age in dog years whose age is three human years.

[Video description begins] *The output reads: Enter dog's age in human years: 3 The dog's age in dog years is 25.* [Video description ends]

The first if and the second if evaluates to false, we'll move into the else block. For the first two years the dog's age is 21, the remaining one year will be multiplied by 4, the dog's age is 21 plus 4, 25. Let's write a Python program to check whether an alphabet is a vowel or a consonant.

[Video description begins] *In a cell, she enters the code, code starts: letter = input("Enter a letter of the alphabet: ") letter = letter.lower() if letter in ('a', 'e', 'i', 'o', 'u'): print("%s is a vowel." % letter) elif letter == 'y': print("Y is ambiguous. It depends where it is used") else: print("%s is a consonant." % letter). Code ends.* [Video description ends]

We'll accept an input from the user, the input should be an alphabet. To make things simple, we'll just convert the letter into a lowercase. The first if condition checks whether the letter is present in the tuple of vowels containing a, e, i, o, u. If this evaluates to true, the letter is a vowel is printed out to screen. If this evaluates to false, we'll move into the elif block, where the condition will check if the letter is equal to y. If this condition evaluates to true, y is ambiguous, it depends where it is used will be printed out to screen. If the elif condition also evaluates to false, we'll move into the else block, the letter is a consonant will be printed out to screen. Let's check for the letter o.

[Video description begins] *The output reads: Enter a letter of the alphabet: o o is a vowel.* [Video description ends]

o is present in the tuple of vowels, the if statement evaluates to true and o is a vowel is printed out to screen. Let's check for another letter, the letter i, and as expected, i is a vowel is printed out to screen.

[Video description begins] *The output reads: Enter a letter of the alphabet: i i is a vowel.* [Video description ends]

Let's check for the letter y, the first if statement evaluates to false.

[Video description begins] *The output reads: Enter a letter of the alphabet: y Y is ambiguous. It depends where it is used.* [Video description ends]

We move into the elif block, the elif block evaluates to true. Y is ambiguous, it depends where it is used is printed out to screen. Let's write code that will accept an input from the user.

[Video description begins] *In a cell, she enters the code, code starts: month_name = input("Enter the name of the Month: ") if month_name == "February": print("No. of days: 28/29 days") elif month_name in ("April",* *"June", "September", "November"): print("No. of days: 30 days") elif month_name in ("January", "March", "May", "July", "August", "October", "December"): print("No. of days: 31 days") else: print("Give a correct month name"). Code ends.* [Video description ends]

The user input should provide the name of the month, and will store it in the variable month_name. The code will check the number of days present in the given month. The first if condition checks if the month name is February, then the number of days are 28 or 29. If this evaluates to false we'll move into the elif block. The elif condition checks if the month name is present in the tuple of April, June, September, November, then the number of days are 30 days. If this evaluates to false we'll move into the next elif block. Where the condition will check if the month name is present in the tuple of January, March, May, July, August, October, December. Then, the number of days are 31 days. If the second elif condition also evaluates to false, we'll move into the else block and give a correct month name will be printed out to screen. Let's check how many days are there in the month of May.

[Video description begins] *The output reads: Enter the name of the Month: May No. of days: 31 days.* [Video description ends]

The number of days are 31. Let's check the number of days for the month of April.

[Video description begins] *The output reads: Enter the name of the Month: April No. of days: 30 days.* [Video description ends]

Number of days are 30 days.

[Video description begins] *In a cell, she enters the code, code starts: a = int(input("a: ")) b = int(input("b: ")) c = int(input("c: ")) if a == b == c: print("Equilateral triangle") elif a == b or b == c or c == a: print("Isosceles triangle") else: print("\nScalene triangle"). Code ends.* [Video description ends]

Let's write a program to check whether a triangle is an equilateral triangle, an isosceles triangle, or a scalene triangle. We'll accept three inputs from the user, a, b, and c each of them being the length of a side of a triangle. The first if statement evaluates if a is equal to b is equal to c, that is if all the sides are equal, then it is an equilateral triangle. If this evaluates to false we'll move into the elif block. We'll check if any of the two sides are equal. a == b or b == c or c == a. If this evaluates to true, it's an isosceles triangle. Remember here that the first if check has to be to see whether all three sides are the same. This is because the elif here will be true for both equilateral and isosceles triangle. If the elif condition also valids to false, we'll move into the else block, and we'll know that our triangle is a scalene triangle. Execute this cell, a is 12, b is 12, and c is 10. The length a and the length b are equal, the elif evaluates to true.

[Video description begins] *The output reads: a: 12 b: 12 c: 10 Isosceles triangle.* [Video description ends]

This is an isosceles triangle. Let's check for another triangle, a is 10, b is 12, c is 14, none of the sides are equal.

[Video description begins] *The output reads: a: 10 b: 12 c: 14 Scalene triangle.* [Video description ends]

It's a scalene triangle.

[Video description begins] *In a cell, she enters the code, code starts: str_value = input("Enter the string: ") reversed_str = str_value[:: -1] if(str_value == reversed_str): print("The string is a palindrome") else: print("The string is not a palindrome"). Code ends.* [Video description ends]

Let's accept an input from the user and check whether the string is a palindrome or not. A palindrome is a word or a sequence that reads the same as backwards as forward. We'll store the user input in the string value variable. We'll reverse the string value by using the list slicing operation and store it in the reverse string variable. Remember that strings are a list of characters. The step parameter is -1 here, that means it will reverse the whole string. The if statement evaluates if the string value is equal to the reverse string. Then the string is a palindrome else the string is not a palindrome. Let's check if the string python is a palindrome.

[Video description begins] *The output reads: Enter the string: python The string is not a palindrome*. [Video description ends]

And as expected, the string is not a palindrome.

[Video description begins] *The output reads: Enter the string: madam The string is a palindrome*. [Video description ends]

Let's check for the string madam, if condition evaluates to true the string is a palindrome.

[Video description begins] *In a cell, she enters the code, code starts: ans = int(input("How many days are there in a leap year? ")) print("You entered:", ans) if ans == 366 : print("You have cleared the first level.") ans = input("\nWhich month has an extra day in a leap year?").lower() if ans == "february" : print("You have cleared the test.") else : print("You have failed the test.") else : print("Your answer is wrong, please try again."). Code ends*. [Video description ends]

Let's write a program which will ask the user how many days are there in a leap year. If the user correctly answers the question, the program will ask which month has an extra day in a leap year? The answer variable will store the answer of the user for the question, how many days are there in a leap year? If the answer is 366, you have cleared the first level will be printed out to screen. Then the user will be asked another question, which month has an extra day in a leap year? The input from the user will be stored in the answer variable. If the user input is February, you have cleared the test will be printed out to screen. If this block evaluates to false, we'll move into the else block, you have failed the test will be printed out to screen. Execute this cell. The first question is correctly answered by the user.

[Video description begins] *The output reads: How many days are there in a leap year? 366 You entered: 366 You have cleared the first level. Which month has an extra day in a leap year?march You have failed the test*. [Video description ends]

You have cleared the first level is printed out to screen. Then the user is asked, which month has an extra day in a leap year? The user input is March. The nested if block evaluates to false. We'll move into the nested else block, you have failed the test is printed out to screen. The user takes the test again.

[Video description begins] *The output reads: How many days are there in a leap year? 366 You entered: 366 You have cleared the first level. Which month has an extra day in a leap year?february You have cleared the test*. [Video description ends]

Now both the questions are correctly answered, you have cleared the test is printed out to screen. Let's write code to formulate a grading system according to the scores achieved by the students.

[Video description begins] *In a cell, she enters the code, code starts: score = int(input("Enter a score: ")) if score >= 90: print("A grade") elif score >=80: print("B grade") elif score >=70: print("C grade") elif score >= 50: print("D grade") else: print("Fail"). Code ends*. [Video description ends]

If the score is more than or equal to 90, the student will get an A grade. If the score is more than or equal to 80, the student will get a B grade. Notice that the first check has to be more than or equal to 90, because the elif condition will be true for all the scores that are above 90 and 80. So all the checks have to be from the highest score to the lowest score. If this score is more than or equal to 70, the student will get a C grade. If the score is more than or equal to 50, the student will get a D grade. If the score is less than 50, the student will fail. Let's check the grade of the student scoring 75, he will get a C grade.

[Video description begins] *The output reads: Enter a score: 75 C grade*. [Video description ends]

A company has decided to give out bonus to their employees.

[Video description begins] *In a cell, she enters the code, code starts: salary = int(input("Enter annual salary: ")) years_service = int(input("Enter years of service: ")) if years_service >= 10: print ("Bonus is: ", .15 * salary) elif years_service >= 5 and years_service < 10: print ("Bonus is: ", .05 * salary) else: print ("No bonus"). Code ends*. [Video description ends]

If the employee has worked more than or equal to 10 years for that company, he will get 15% of his salary as bonus. If the employee has worked between 5 to 10 years, he will get 5% of his salary as bonus. We'll write code to find the bonus amount for an employee. We'll accept two inputs from the user, the salary and the years of service. The if condition evaluates if the years of service is more than or equal to 10, then we'll calculate the bonus as 15% of the salary. If this evaluates to false, we'll move into the elif block. The elif condition evaluates if the years of service is between 5 to 10 years, then the salary will be multiplied by 5% to calculate the bonus. If this condition also evaluates to false, we'll move into the else block, and no bonus will be printed out to screen. Let's calculate the bonus amount for an employee having a salary of $40,000. The years of service is 8 years, the bonus amount is $2,000.

[Video description begins] *The output reads: Enter annual salary: 40000 Enter years of service: 8 Bonus is: 2000.0*. [Video description ends]

Exercise: If-else Statements in Python

[Video description begins] *Topic title: Exercise: If-else Statements in Python. The presenter is Yukti Kalra*. [Video description ends]

In this exercise, you will start off with changing the order of precedence of operators in an expression. In this course, we studied in detail how the order of presidents of operators can affect the evaluation of an expression. Here we have an expression 10 - 4 x 2. The multiplication symbol has higher precedence than the subtraction symbol. You will have to find a way such that the subtraction is carried out first and then the multiplication. Following that, you will summarize the membership operators in Python. These operators are used to validate the membership of a value. For example, you have a list of colors pink, blue, and red. You have to check whether the elements pink and purple are present in the list or not. After this, you will convert a string to a list of characters. As discussed earlier, Python has built-in functions to convert between data types. You will need to convert the string hello into a list of characters containing elements H-E-L-L-O. Each of these topics were covered during this course. So please pause this video and spend some time to do this exercise on your own.

[Video description begins] *Changing the Order of Precedence* [Video description ends]

We'll begin with the first task in this exercise, where you needed to change the order of precedence of operators.

[Video description begins] *Solution* [Video description ends]

You want to execute this expression, 10 - 4 x 2, such that (10- 4) is evaluated first, and then the result is multiplied by 2. The expression should evaluate to 12 instead of 2. The parenthesis has the highest precedence. If we enclose the subtraction expression within parentheses, the subtraction will be carried out first, 10 - 4 is 6 multiplied by 2, gives out the result 12.

[Video description begins] *Membership Operators* [Video description ends]

For the next task in the exercise, you had to summarize the membership operators.

[Video description begins] *The code, colors = ["pink" , "blue" , "red"] is displayed*. [Video description ends]

You had to check if the element pink is present in the list of colors, and if the color purple is present in the list of colors. First, let's work with the in membership operator. The in operator is used to check if a value exists in a sequence or not. It evaluates to true if it finds the value in the specified sequence. Otherwise, it evaluates to false. To check whether the element pink is present in the list of colors or not, we'll use an if statement. If pink in colors, print("this color is present in the list").

[Video description begins] *The code, code starts: if "pink" in colors: print("this color is present in the list"), code ends is displayed.* [Video description ends]

The if statement will evaluate to true, this color is present in the list will be printed out to screen. Now to check whether the color purple is present in the list or not, let's use the not in membership operator. The not in operator evaluates to true if it does not find the value in the specified sequence. Otherwise, it evaluates to false. We'll use the if statement again to check whether the element purple is present in the list of colors or not. The if statement evaluates if "purple" not in colors. If this evaluates to true, then print("this color is not present in the list").

[Video description begins] *The code, code starts: if "purple" not in colors: print("this color is not present in the list"), code ends is displayed.* [Video description ends]

The color purple is not present in the list of colors. Hence, the if statement evaluates to true. This color is not present in the list will be printed out to screen.

[Video description begins] *Convert a String to a List* [Video description ends]

Now let's move on to the third task in this exercise where you had to convert a string into a list of characters.

[Video description begins] *The code, my_string = "hello" is displayed.* [Video description ends]

In Python, strings are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with the length of one. Here we have a variable my_string initialized with the string "hello". We'll invoke the list function on the my_string variable and store it in the my_list variable.

[Video description begins] *The code, my_list = list(my_string) is displayed.* [Video description ends]

The list constructor creates a list in Python. The list function dates sequence types and converts them to lists. Let's print the my_list output screen.

[Video description begins] *The code, print(my_list) is displayed.* [Video description ends]

We get a list of characters. The elements are 'h','e', 'l', 'l', 'o'.

[Video description begins] *The output ['h', 'e', 'l', 'l', 'o'] is displayed.* [Video description ends]