Hi, welcome to Basic Program Structure.

☼ More on Selection (Branching) in Python

☼ **At the end of this lesson, you should be able to:**

Apply the following selection (branching) structures in Python:

- IF-ELSE statement
- IF-ELIF-ELSE statement
- Nested IF statement

At the same time, **you should be able to explain all the basic Arithmetic operators in Python and use them to solve problems.**

☼ In this lesson, we will go through them in details one by one.

☼Let's recall the if-else structure we have learnt in the previous lesson.

Evaluate condition, that is, a Boolean expression to yield True or False.

If the condition yields True,

then execute suite A

and continue with rest of the program.

If the condition yields False,

then execute suite B,

and continue with rest of the program.

☼What are the syntax rules of if-else structure in Python?

The if-else statement has two compound statements, These statement blocks can have any number and type of statements.

with two headers: the if and the else.

Pay particular attention to the indentation: the else and the if part of the expression are at the same indentation (they are part of the same overall statement). The if has an associated Boolean expression; the else does not. There is a colon behind if or else.

☼In Python, a colon marks the start of a block or suite. If we omit the colon, what will happen? Syntax error. Do remember, do not forget the colon. Otherwise, it causes a syntax error!

☼ Here's a familiar example. Read in the radius of a circle and calculate circumference and area. With the help of if-else statement, we can do error checking Negative radius will be blocked and the meaningful error message will be displayed to the user.

☼One more interesting application of if-else statement. Determine **if an integer is divisible by 3. The flow chart is easy to understand. But when we think about the detailed implementation.**

☼**How to determine whether m is divisible by n? That is, How to determine whether the remainder of m/n is 0, not in math, but in Python. We need introduce more Arithmetic operators in Python to enhance our ability to solve real problems.**

☼**Modulus operator returns the remainder of the division of left operand by the right.** For example, if x equals to 8, y equals to 3, 8 divided by 3, the remainder is 2. So, 8 % 3 returns 2.

Here is a simple sample code to determine if n is the divisor of m. What is a divisor ? For a number m, a divisor is a number that divides into m equally. It means that the remainder is zero.

☼Here's another interesting **Application of Modulus operator : today is Tuesday, then after 53 days, what day will it be?**

Answer is Saturday. Why? A cycle of a week is 7 days. **We divide 53 by 7,**

```
 7
7)53
 49
 4
```

So that's 7 full weeks and the remainder 4 tells us it's 4 days over 7 weeks, and 4 days af ter Tuesday is Saturday.

☼All these **Arithmetic operators can work for both floating points and integer numbers. Floating point operators work as expected, so we will focus on integer operators.**

☼**As we have learnt in previous part, most of the operators, such as addition, subtraction, and multiplication, of integers work exactly as you learned in primary school arithmetic class.**

☼**Division works as you would expect as well, but it is a little different from the other operators.**

☼If you add two integers, the sum is an integer. The same is true for subtraction and multiplication. However, if you divide an integer by another integer , the result is not necessarily an integer. It could be an integer (say, 8 divided by 4) or it could be a float (say 8/3). Mathematically, it is a fraction, Python represents a fraction as a float. For consistency, whenever you do division, regardless of the type of the operands, the type yielded is a float. You have to convert it to an integer if necessary.

However, perhaps indeed you only want an integer result. Python provides an operator that when used with integers provides only the integer part of a division, that is, the quotient. That operator is floor division operator.

☼Note that Division by Zero will cause run time error.

☼Quick check: **Grace has $25. She wants to buy movie tickets at $3 each. Which of the following statements calculates the maximum number of tickets she can purchase?**

☼The answer is C. Why? **The number of the movie ticket must be an integer. Option A returns a float while option D is not a valid operator. So we only need look at options B and C. B is modulus operator to return the remainder and C is the floor division to return the quotient. The quotient represents the maximum number of tickets she can purchase. So, C is the correct answer.**

☼What is the difference between the number 8 and 8.0? The answer is that they are of different types. They have the same value, but they are different in type. Therefore, when you perform operations with them, you might get different answers.

What happens when you have mixed types? For example, how does the computer handle dividing an integer by a floating-point number? The answer is, "it depends." In general, operations provided by each type dictate /dɪ[k]ˈteɪt/ **the rules of what can and cannot be mixed.**

For numbers, a computer has a separate hardware for integer and floating-point arithmetic. Because of that hardware, the operation is best done in one type or the other, so the numbers must be of the same type. Those that are not of the same type must be converted.

Given that, what is the correct conversion to perform: integer-to-float or float-to-integer?

Clearly no information is lost when converting an integer to a float, but conversion of a float to an integer would lose the fractional information in the float. Therefore, most programming languages, including Python, when presented with mixed types will "promote" an integer to be a floating point so that both operands are floats and the operation can be performed as floats.

☼The order of arithmetic operations in Python and most common programming languages is the same as the one you learned in arithmetic: multiplication and division before addition or subtraction. The term used to describe the ordering is precedence . In this case, we say that multiplication and division have greater precedence than addition or subtraction, so they are done first. Further, exponents have greater precedence than multiplication and division, also as in arithmetic.

If operations have the same precedence —for example, multiplication and division—they are done left to right. Finally, as in arithmetic, parentheses or round brackets can be used to

override the precedence order and force some operations to be done before others, regardless of precedence.

The precedence of arithmetic operations is shown in the Table.

☼Operations—especially groups of operations—that are used repeatedly are often provided with a shortcut, a simpler way of typing. Python, like other languages, has such shortcuts.

They are not required, but they do make typing easier and, at some point, make the code shorter and easier to read.

Listed here, they are a combination of one of the arithmetic operators, such as + plus, -minus, *times, /divides, with the assignment sign = .

Some examples would be +=, -=, /=, *=. Note that the operation comes before the

assignment sign, not after!

What does += mean? Let's look at an example: myInt += 2. The augmented operator, in general, means "Perform the augmented operation (plus here) using the two operands, the value 2 (value on the right side) and myInt (variable on the left side), and reassign the result to myInt (variable on the left side)". That is, the following two expressions are exactly equivalent: myInt += 2 and myInt = myInt + 2. The first is the shortcut, the second the "long" way.

☼**Quick check: What is the output of the following code?**

☼Answer is C.

X is assigned a float value of 1.5

Y is assigned an integer value of 2

X * = y is the shortcut of x = x * y

Two operands of multiplication with different data types, must be converted to be same. Integer is converted to float. 1.5*2.0 = 3.0

So the answer is C 3.0

☼ Besides a**ugmented Assignments,** Python has an interesting and unique operator, chained relational operator to make the code shorter and easier to read.

Chained relational expressions work just like they do in mathematics (which is not true for many programming languages):

For example myInt has a value of 5. **The expression 0 <= myInt <= 5 represents is** an integer greater than or equal to 0 and less than or equal to 5?It is equivalent to **0 <= myInt and myInt <= 5, but the short form is closer to natural** mathematics expression. The expression is converted into two relations, which are and'ed together. You can then evaluate each relational expression and combine the results referring to the and table.

There are a lot of useful applications of Chained relational expressions. for example, salary range, marks range.

Let's look at a more complex one. 0 <= myInt <= 5 > 10. What will be the output? False.

Just apply each operator to compare its two **neighbouring** values and then "and" the results.

0 <= myInt and myInt <= 5 and 5 > 10

True and True and False, that is True and False, finally, fields False

☼We have learned that the precedence of arithmetic Python operators is the same as in arithmetic. We have learned relational operators in previous lesson. Where do they fit in precedence with respect to the arithmetic operators? The Table shows the combined precedence. Now we have solid foundation to move on.

☼A third kind of header available to us is the elif header. The elif is simply shorthand for "else if". The elif header has an associated condition, just like an if statement. It too has an associated suite.

The basic idea of a set of if-elif-else statement is to find the first True condition

in the set of if and elif headers and execute that associated suite. If no such True

condition is found, the else is executed. The important point is that, only one suite will be executed.

The **if-elif-else** statement operation is as follows
. Evaluate boolean expression1 to yield True or False.
2. If boolean expression1 yields True,
(a) Execute suite1.
(b) Continue with the next statement after suite last.
3. If boolean expression1 yields False, then evaluate boolean expression2.
If boolean expression2 yields True,
(a) Execute suite2.
(b) Continue with the next statement after suite last.
4. If all preceding boolean expressions yield False,
(a) Execute the **else** part and suite last.
(b) Continue with the next statement after suite last.

☼Again, note the indentation: the keywords (if, elif, else) are indented the same amount as they are part of the overall if statement. Further, each of the if, elif, and else statements can have an associated suite.

☼You may have many, many ELIF blocks. all if, elif, and else statements form a Whole statement. For each elif statement the associated suite will be executed if all preceding boolean

expressions are False and its boolean expression is True. If all if and elif conditions are False, then the else clause is executed.

☼It is worth examining the flow of control in a little more details using the following program.

When the code is evaluated with mark set to 95, the if condition is evaluated, returning True. Note that none of the other conditions is evaluated. When the code is evaluated with mark set to 55, all of the if and elif conditions are evaluated, returning False, thus requiring the else suite to be executed.

☼quick check: Examine the same situation in different program. Is it correct?

☼No.

Take the example of having mark set to 95. The output of second program is "Congratulations. You received an A. oops, not good. Wish you can be better next time." Ridiculous message, there must be some logic error.

☼Let's analyse the two programs.

Note that all the elif headers are replaced with if headers. The suites are unchanged.

First, how many conditions will be evaluated in second program if mark is set to 95? The answer is, all of the 5 conditions will be evaluated, even though the first one would return True. This is because, in the second program, there are five statements.

In the first program, there is only one if statement. The combination of if, elif, and else constitutes one logical statement.

Second, in second program, with what statement is the final else associated ? An else is associated with the most recent if statement at the same indentation level, in this case it is "**if 60 <= mark < 70:**" .

Each if can have one else and each else must have an associated if. It is for this reason that we get the "oops" output for the input of 95. Make sure you can follow the flow of control for this example.

You are free to mix and match, but make sure you understand how that combination works.

☼We have mentioned nesting in previous lesson. It is worth to have a little more discussion on it

Syntax wise, we have learned all the headers in selection structure, and NO new headers are required in nested if structure.

☼It is easy for us to understand the logic of the program. Note that each time we insert another control construct, its suite will be indented. In this program, proper indentation is needed for the second level.

☼The indentation aids readability, but if the code is nested too much, the indentation will be difficult to read. So, in reality, usually, there is a limit of nesting level. If you reach it, your code is unreadable and should be restructured. One tool, which we will cover later, is a function, which allows us to improve readability by encapsulating some of the indentation within a function.

☼It is time for us to start the coding of **Scenario 6. Looking at the flow chart, we have second level selection along both true path and false path. So both nested if –else and if-elif-else structures will be used.**

☼**The suggested code is shown in this slide.**

☼Quick summary. The if statement expresses selective execution in Python, but it comes in a number of variations. We have learnt them in this lesson. Syntax wise, please Pay particular attention to the indentation and colon.

☼We have introduced all the commonly used **Relational and Arithmetic Operators. With these concepts, you now have sufficient programming power to write powerful programs.**