HyperionDev

Logical Programming – Operators

Task

Visit our website

Introduction

In a programming language, an operator is a symbol that tells the compiler or interpreter to perform specific operations (whether it be mathematical, relational, or logical) and produce a final result. This task will introduce you to the different types of operators and show you how to use them.

Types of operators

Operators are symbols that tell the computer which mathematical calculations to perform or which comparisons to make.

Comparison operators

Here is a quick look at four basic comparative operators:

- greater than >
- less than <
- equal to ==
- not !

We can combine the greater than, less than, and not operator with the equals operator to form three new operators:

- greater than or equal to >=
- less than or equal to <=
- not equal to !=

You can use == to check if two strings are identical, for example:

```
my_name = "Tom"

if my_name == "Tom":

    print("I was looking for you")
```

Comparative operations are especially useful for comparing numbers:

```
num1 = 10

num2 = 20

if num1 >= num2:  # The symbol for “greater than or equal to” is >=

    print("It's not possible that 10 is bigger than or equal to 20.")

elif num1 <= num2:  # The symbol for “less than or equal to” is <=

    print("10 is less than or equal to 20.")

elif num1 != num2:  # The symbol for “not equal to” is !=

     print("This is also true since 10 isn't equal to 20, but the elif statement before comes first and is true so Python will execute that and never get to this one!")

elif num1 == num2:  # The symbol for “equal to” is ==

    print("Will never execute this print statement...")
```

The program will check the first part of the `if` statement (is `num1` bigger than or equal to `num2`?).

If it isn’t, then it goes into the first `elif` statement and checks if `num1` is less than or equal to `num2`. If it isn’t, then it goes into the next `elif` statement, etc. In the example above, the first `elif` will execute.

Logical operators

A logical operator is another type of operator that is used to control the flow of a program. Logical operators are usually found as part of a control statement, such as an `if` statement. Logical operators basically allow a program to make a decision based on multiple conditions; for instance, if you would like to test more than one condition in an `if` statement.

The three logical operators are:

Operator	Explanation
`and`	Both conditions need to be true for the whole statement to be true (also called the conjunction operation).
`or`	At least one condition needs to be true for the whole statement to be true (also called the disjunction operation).
`not`	The statement is True if the condition is False (only requires one condition).

The `and` and `or` operators require two operands, while the `not` operator requires one.

Let's take a real-life situation. When buying items at a store, two criteria need to be met: The item needs to be in stock and you need to have enough money to pay for it. This is an instance of a conjunction operation where both conditions needs to be True for the whole statement to be True. This would be represented using the `and` operation.

Let's look at another situation. A person could receive a good mark at school either because they are very bright or because they studied hard. In this instance, either one of the options can be True, or both can be True, but at least one needs to be True. This is a disjunction operation, where at least one of the conditions need to be met for the whole statement to be True. This would be represented using the `or` operator.

Example of an `and` operator:

```
team1_score = 3

team2_score = 2

game = "Over"

if (team1_score > team2_score) and (game == "Over"):

    print("Congratulations Team 1, you have won the match!")
```

Example of an `or` operator:

```
speed = int(input("How many kilometres per hour are you travelling at?"))

belt = input("Are you wearing a safety belt?")

if (speed > 80) or (belt != "Yes"):

    print("Sorry Sir, but I have to give you a traffic fine.")
```

Arithmetic operators

The arithmetic operators in Python are as follows:

Arithmetic operators	Python Symbol
Addition: adds values on either side of the operator.	**+**
Subtraction: subtracts the value on the right of the operator from the value on the left.	**–**
Multiplication: multiplies values on either side of the operator.	*****
Division: divides the value on the left of the operator by the value on the right.	**/**
Modulus: divides the value on the left of the operator by the value on the right and returns the remainder. E.g., `4 % 2 == 0` but `5 % 2 == 1`.	**%**
Exponent: performs an exponential calculation, i.e., calculates the answer of the value on the left to the power of the value on the right.	******
Assignment operators	**Python Symbol**
Equals: sets the value of what is on the left to that of what is on the right. E.g., `n = 7`	**=**
Plus-equals: adds `1` to the variable for each iteration. E.g., `n += 1` is shorthand for `n = n + 1`. (This is particularly useful when using loops, as you will eventually see.)	**+=**
Minus-equals: minuses `1` from the variable for each iteration. E.g., `n -= 1` is shorthand for `n = n - 1`.	**–=**

Instructions

First, read and run the **example file** provided. Feel free to write and run your own example code before doing the practical task to become more comfortable with the concepts covered in this task.

Take note

The task below is **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request review”, the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

Auto-graded task

Follow these steps:

- Create a new Python file in the task folder called **award.py.**
- Design a program that determines the award a person competing in a **triathlon** will receive.
- Your program should accept user inputs for the times (in minutes) of all three triathlon events, namely swimming, cycling, and running, and then calculate and output the total time to complete the triathlon.

    - For example: `Total time taken for the triathlon: 82 minutes`

- The award a participant receives is based on the total time taken to complete the triathlon. Determine the award that the participant will receive based on the following criteria:

Qualifying criteria	**Time range**	**Award**
Within the qualifying time.	0–100 minutes	Provincial colours
Five minutes off from the qualifying time.	101–105 minutes	Provincial half colours
10 minutes off from the qualifying time.	106–110 minutes	Provincial scroll
More than 10 minutes off from the qualifying time.	111+ minutes	No award

- Output the award they will receive or 'No award'.
    - For example: `Award: Provincial scroll`

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.