DataScientest • com

Python for Data Science

Operators and Control Structures

Introduction

Operators such as `+` or `=` are used to perform operations on variables and on their values. These operators belong to the large family of Python operators. In this exercise, we will learn how to handle the following categories of operators:

- Arithmetic operators.
- Assignment operators.
- Comparison operators.
- Membership operators.
- Logical operators.

We will then see how to use these operators to control the execution of code using the `if`, `else` and `elif` statements.

1. Arithmetic Operators

The **arithmetic operators** applicable to **numeric** variables are as follows:

Symbol	Operation	Example
`+`	Addition	`6+4` returns `10`
`-`	Subtraction	`6-4` returns `2`
`*`	Multiplication	`6*4` returns `24`
`/`	Real division	`6 / 4` returns `1.5`
`//`	Floor division	`6.0 // 4` returns `1`
`**`	Exponentiation	`6 ** 4` returns `1296`
`%`	Modulus	`6 % 4` returns `2`

Integer or Floor division `//` reflects how many times the number on the right can fully divide the number on the left. For example, `7 // 2` is equal to 3 because 2 can fit in 7 three times.

The **modulus** operator `%` computes the **remainder** of the floor division between 2 numbers. For example, `7 % 2` is 1 because 7 is divisible by 2 three times and what remains is 1.

The `%` operator is used to determine whether a number is odd or even. Indeed, if a number `n` is even, then necessarily `n% 2` is equal to 0. Similarly, if `n` is odd, then `n% 2` is equal to 1.

- **(a)** Create the variable **`distance`** and assign it the value `750` (distance between Paris and Marseille in km).

In [1]:

```python
# Insert your code here

distance = 750
speed = 4.8

time = distance / speed

day = time // 24

hours = time % 24

print(f"The walker would need {day} days and {hours} hours.")
```

```
The walker would need 6.0 days and 12.25 hours.
```

Hide solution

In [2]:

```
# Distance in km between Paris and Marseille
distance = 750

# Average speed of a walker in km/h
speed = 4.8

# Time in hours it would take to go from Paris to Marseille without stopping
time = distance/speed

# Number of days of walking
days = time // 24

# Number of hours remaining
hours_remaining = time % 24


print("The walker would need", days, "days and", hours_remaining, "hours.")
```

```
The walker would need 6.0 days and 12.25 hours.
```

2. Assignment Operators

For all arithmetic operations, such as addition or multiplication, there is a way to apply both the operation and assignment at once through the operators ' += ' or ' *= ' for example.

Symbol	Operation
+=	Addition
−=	Subtraction
*=	Multiplication
/=	Real division
//=	Floor division
**=	Exponentiation
%=	Modulus

So `x += 3` is **equivalent** to `x = x + 3`. Similarly, `z**= 2` is equivalent to writing `z = z**2`.

A magician claims that for any **prime number** different from 2 and 3, if:

- We **square** it.
- We **add 17** to it.
- We **divide it by 12** and we keep the **remainder** of this division, then the remainder of this division is **6**.

- **(a)** Is he right?

Remember that a prime number is a number which is only divisible by 1 or itself. The 10 smallest prime numbers other than 2 and 3 are 5, 7, 11, 13, 17, 19, 23, 29, 31, 37.

In [3]:

```
prime_numbers = [5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
x = prime_numbers[0]

# Insert your code here
for x in prime_numbers:
    x **= 2
    x += 17
    x %= 12
    print(x)
```

```
6
6
6
6
6
6
6
6
6
6
```

Hide solution

In [ ]:

```
prime_numbers = [5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
x = prime_numbers[0]

# We square x
x **= 2

# We add 17 to x
x += 17

# We keep the remainder of the floor division of x by 12
x %= 12

print(x)
# The magician is right
```

3. Comparison Operators

The comparison operators allow you to compare the values of two variables. Comparison operations return the Boolean value `True` if the expression is true, or `False` if it turns out to be false. For example:

```
x, y = 3, 5

# Is x smaller than y?
print (x < y)
>>> True
```

Python's comparison operators are:

Expression	Example	Meaning
<	x < y	Is x **less than** y ?
<=	x <= y	Is x **lesser or equal** to y ?
>	x > y	Is x **greater** than y ?
>=	x >= y	Is x **greater or equal** to y ?
==	x == y	Is x **equal** to y ?
!=	x != y	Is x **different** from y ?

Important Example:

```
x, y = 3, 5

# Is x equal to y?
print (x == y)
>>> False
```

YOU MUST NOT CONFUSE `x == y` with `x = y`. The first instruction is a **comparison** operation while the second one is an **assignment**.

- **(a)** In a single line of code, determine with a boolean value if 7 divides $3^7 + 2^{14}$.

As on a calculator, you can use **parentheses** to set operation priorities.

In [8]:

```
# Insert your code here

# a)
print((3**7+2**14) % 7 == 0)

# b)
a_list = [4, 5, 10]
for n in a_list:
    print((3**(2*n+1)+2**(4*n+2)) % 7 == 0 , n)
```

```
True
True 4
True 5
True 10
```

Hide solution

In [7]:

```
# First question
print((3**7 + 2**14) % 7 == 0)

# Second question
n = 4
x = 3**(2*n + 1) + 2**(4*n + 2)
print(x % 7 == 0)

n = 5
x = 3**(2*n + 1) + 2**(4*n + 2)
print(x%7 == 0)

n = 10
x = 3**(2*n + 1) + 2**(4*n + 2)
print(x%7 == 0)
```

```
True
True
True
True
```

4. Membership Operators

Membership operators are used to test whether a value is **absent or present** in a sequence such as a list or a tuple. The operator which determines whether a value is present in a sequence is the operator `in` and the operator which determines whether a value is absent is `not in` :

```
a_list = [1, 3, 102, 32, 11, -12, 33]
x = 14

# Is the value of x one of the values in a_list?
print (x in a_list)
>>> False

# Is the value of x NOT one of the values in a_list?
print (x not in a_list)
```

In [9]:

```
excerpt = ['The', '21', 'World', 'Cup', 'tournaments', 'have', 'been', 'won', 'by', 'eight',
           'national', 'teams.', 'Brazil', 'have', 'won', 'five', 'times', ',', 'and',
           'they', 'are', 'the', 'only', 'team', 'to', 'have', 'played', 'in', 'every',
           'tournament', '.', 'The', 'other', 'World', 'Cup', 'winners', 'are', 'Germany',
           'and', 'Italy', ',', 'with', 'four', 'titles', 'each', ';', 'Argentina', ',',
           'France', ',', 'and', 'inaugural', 'winner', 'Uruguay,', 'with', 'two', 'titles',
           'each', ':and', 'England', 'and', 'Spain', '.', 'with', 'one', 'title', 'each', '.']
```

- **(b)** In a single line of code, determine whether the country of `"France"` is mentioned in this excerpt.
- **(c)** Unfortunately, losers are quickly forgotten even if their performance was historic. Check that the country of `"Croatia"` is not mentioned in the excerpt.

In [10]:

```
# Insert your code here
# b)
print("France" in excerpt)

# c)
print("Croatia" not in excerpt)
```

```
True
True
```

Hide solution

In [11]:

```python
# Is France mentionned in the excerpt?
print("France" in excerpt)

# Is Croatia NOT mentionned in this exceprt?
print("Croatie" not in excerpt)
```

```
True
True
```

5. Logical Operators

Logical operators allow you to perform what is called Boolean **arithmetic**. Typically, when we have multiple Boolean expressions, logical operators are used to determine whether:

- **All** expressions are true.
- **At least one** of the expressions is true.

```
x, y = 3, 5

# Is 3 less than 5? True
expression1 = (x < y)

# Is 5 divisible by 3? False
expression2 = (y % x == 0)

# Are both expressions true?
print(expression1 and expression2)
>>> False

# Is at least one of the expressions true?
print (expression1 or expression2)
>>> True
```

The operator `not` returns the **negation** of a boolean expression:

```
x, y = 3, 5

expression = (y % x == 0)

# Is y divisible by x?
print(expression)
>>> False

> # Is y NOT divisible by x?
print(not expression)
>>> True
```

The logical operators are summarized below:

In [13]:

```python
# Insert your code here

list_name = ["Bernadette", "Marc"]
list_seniority = [12, 6]
list_salary = [2400, 1490]

for i,j,k in zip(list_seniority, list_salary, list_name):
    criterion1 = i < 5 and j < 1500
    criterion2 = (5 <= i <= 10) and (1500 <= j <= 2300)
    criterion3 = (i > 10) and (1500 > j or j > 2300)
    print("Can", k, "receive the bonus?", criterion1 or criterion2 or criterion3)
```

```
Can Bernadette receive the bonus? True
Can Marc receive the bonus? False
```

Hide solution

In [12]:

```
# Bernadette
seniority = 12
salary = 2400

# Does Bernadette have less than 5 years of seniority and is her salary is less than 1500€?
criterion1 = seniority < 5 and salary < 1500

# Does Bernadette have between 5 and 10 years of seniority and is her salary between 1500€ and 2300€?
criterion2 = (5 <= seniority <= 10) and (1500 <= salary <= 2300)

# Does Bernadette have more than 10 years of seniority and is her salary less than 1500€ or higher than 2300€?
criterion3 = (seniority > 10) and (1500 > salary or salary > 2300)

# Can Bernadette receive the bonus?
print ("Can Bernadette receive the bonus?", criterion1 or criterion2 or criterion3)


# Marc
seniority = 6
salary = 1490

# Does Marc have less than 5 years of seniority and is his salary is less than 1500€?
criterion1 = seniority < 5 and salary < 1500

# Does Marc have between 5 and 10 years of seniority and is his salary between 1500€ and 2300€?
criterion2 = (5 <= seniority <= 10) and (1500 <= salary <= 2300)

# Does Marc have more than 10 years of seniority and is his salary less than 1500€ or highis than 2300€?
criterion3 = (seniority > 10) and (1500 > salary or salary > 2300)

# Can Marc receive the bonus?
print ("Can Marc receive the bonus?", criterion1 or criterion2 or criterion3)
```

```
Can Bernadette receive the bonus? True
Can Marc receive the bonus? False
```

6. Control Structures

It is often useful to choose whether a block of code should be executed or not.

For example, if you want to automatically credit the account of civil servants eligible for the government bonus, the variables containing their balances must be updated **only** for civil servants who are **eligible** for the bonus.

Control structures are used to execute a block of instructions under conditions. The two main keywords used to condition instructions are:

- **`If`**
- **`else`**

Suppose the **`eligible`** variable is a Boolean variable (i.e. of value `True` or `False`) telling us whether a civil servant is eligible for the bonus and suppose that the variable **`balance`** corresponds to the amount of money they have in their account.

In order to credit the account of civil servants eligible for the bonus, we can structure our code as follows:

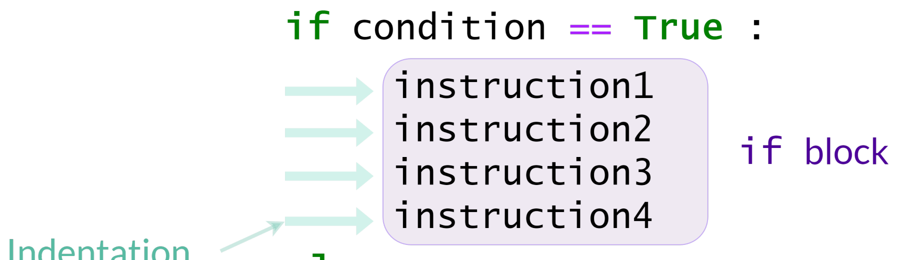
```python
# Is the civil servant eligible for the bonus?
if eligible == True:
    # If so, we increase their balance by 300 euros
    balance += 300
```

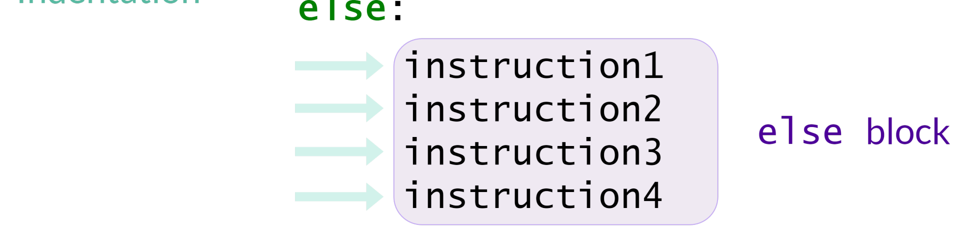
Following numerous complaints, the government will still offer a bonus of **50€** to employees who were **not** eligible for the 300€ bonus.

In order to credit these people's account, we will add an **`else`** statement to our program, whose associated instructions will execute only if `eligible` is equal to `False` :

```python
# Is the civil servant eligible for the bonus?
if eligible == True:
    # If so, we increase their balance by 300 euros
    balance += 300
else:
    # Otherwise, we only increase their balance by 50 euros
    balance += 50
```

The **`:`** character after an `if` or `else` statement allows **to start a block of code**. In order to determine the start and the end of a block, the instructions which will be executed in a block must be **indented**, that is to say shifted by one tabulation or 4 spaces:





An unscrupulous teacher wants to generate student reviews automatically based on their grade. To do this, they can test several conditions one after the other using **`elif`** statements (contraction of `else` and `if`):

```
if grade < 5:
    print("Very insufficient work.")
elif grade < 10:
    print("Could do better.")
elif grade < 15:
    print("Good work. I encourage you to keep it up.")
else:
    print("Excellent work. Congratulations.")
```

In [14]:

```
number = -2

# Insert your code here
if number < 0:
    print("This number is negative.")
elif number == 0:
    print("This number is 0.")
else:
    print("This number is positive.")
```

```
This number is negative.
```

Hide solution

In [ ]:

```
number = -2

if number == 0:
    print ("This number is 0.")
elif number > 0:
    print ("This number is positive.")
else:
    print ("This number is negative.")
```

- **(b)** Is the syntax of the following code correct? If not, suggest a correction.

```
if size < 160:
  print("This person is small.")
else if 160 <= size < 180:
  print("This person is of medium height.")
else 180 <= size < 200:
  print("This person is very tall")
else:
  print("This person is very, very tall")
```

In [19]:

```
size = 205

# Insert your code here

if size < 160:
    print("This person is small.")
elif 160 <= size < 180:
    print("This person is of medium height.")
elif 180 <= size < 200:
    print("This person is very tall")
else:
    print("This person is very, very tall")
```

```
This person is very tall
```

Hide solution

In [ ]:

```python
size = 205

if size <160:
    print ("This person is small.")
elif 160 <= size <180:
    print ("This person is of medium height.")
elif 180 <= size <200:
    print ("This person is very tall.")
else:
    print ("This person is very, very tall.")
```

7. Bonus: Conditional assignment

Our teacher now wants to automatically determine if a pupil is repeating or moving up to the next year. Depending on a student's average grade, the Boolean variable **`repeating`** must take the value `True` if the student's average is less than `10` and `False` otherwise.

As seen previously, we could use `if` and `else` statements:

```python
if average < 10:
    repeating = True
else:
    repeating = False
```

Python allows us to do this operation in one line thanks to a compact and elegant syntax:

```python
redouble = True if average < 10 else False
```

This syntax is strictly **equivalent** to the previous syntax.

Conclusion and Recap

The **arithmetic operators** applicable to numeric type variables are used to perform elementary operations and are summarized below:

Symbol	Operation	Example
`+`	Addition	`6+4` returns `10`
`−`	Subtraction	`6−4` returns `2`
`*`	Multiplication	`6*4` returns `24`
`/`	Real division	`6 / 4` returns `1.5`
`//`	Floor division	`6.0 // 4` returns `1`
`**`	Exponentiation	`6**4` returns `1296`
`%`	Modulus	`6 % 4` returns `2`

For all arithmetic operations, there is a way to apply the operation and assignment all at once through the following **assignment operators**:

Symbol	Operation
`+=`	Addition
`−=`	Subtraction
`*=`	Multiplication
`/=`	Real division
`//=`	Floor division
`**=`	Exponentiation
`%=`	Modulus

Example: `x + = 10` is equivalent to `x = x + 10` .

The **comparison operators** in Python are:

Expression	Example	Meaning
`<`	`x < y`	Is `x` **less than** `y` ?
`<=`	`x <= y`	Is `x` **lesser or equal** to `y` ?
`>`	`x > y`	Is `x` **greater** than `y` ?
`>=`	`x >= y`	Is `x` **greater or equal** to `y` ?
`==`	`x == y`	Is `x` **equal** to `y` ?
`!=`	`x != y`	Is `x` **different** from `y` ?

These operators are mainly used to build control structures thanks to the **`if`** , **`else`** and **`elif`** statements:

```
    if size < 160:
        print ("This person is small.")
    elif 160 <= size <180:
        print ("This person is of medium height.")
    elif 180 <= size <200:
        print ("This person is very tall.")
```

✖ Unvalidate