CONTROL FLOW

Introduction to Control Flow

Imagine waking up in the morning.

You wake up and think, “Ugh. Is it a weekday?”

If so, you have to get up and get dressed and get ready for work or school. If not, you can sleep in a bit longer and catch a couple extra Z’s. But alas, it is a weekday, so you are up and dressed and you go to look outside, “What’s the weather like? Do I need an umbrella?”

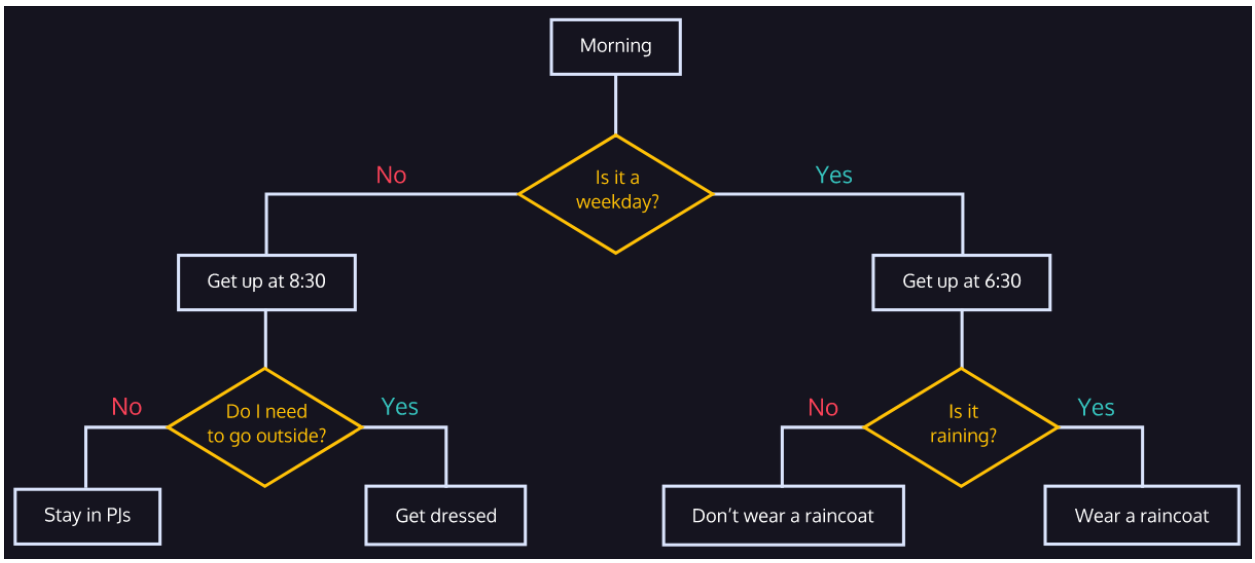
These questions and decisions control the flow of your morning, each step and result is a product of the conditions of the day and your surroundings. Your computer, just like you, goes through a similar flow every time it executes code. A program will run (wake up) and start moving through its checklists, is this condition met, is that condition met, okay let’s execute this code and return that value.

This is the *control flow* of your program. In Python, your script will execute from the top down, until there is nothing left to run. It is your job to include gateways, known as conditional statements, to tell the computer when it should execute certain blocks of code. If these conditions are met, then run this function.

Over the course of this lesson, you will learn how to build conditional statements using boolean expressions, and manage the control flow in your code.

Instructions

Click **Next** to proceed to the next exercise.



Boolean Expressions

In order to build control flow into our program, we want to be able to check if something is true or not. A boolean expression is a statement that can either be `True` or `False`.

Let's go back to the 'waking up' example. The first question, "Is today a weekday?" can be written as a boolean expression:

```
Today is a weekday.
```

This expression can be `True` if today is Tuesday, or it can be `False` if today is Saturday. There are no other options.

Consider the phrase:

```
Friday is the best day of the week.
```

Is this a boolean expression?

No, this statement is an opinion and is not objectively `True` or `False`. Someone else might say that "Wednesday is the best weekday," and their statement would be no less `True` or `False` than the one above since there is no objective answer to what the best day of the week is.

How about the phrase:

```
Sunday starts with the letter 'C'.
```

Is this a boolean expression?

Yes! This expression can only be `True` or `False`, which makes it a boolean expression. Even though the statement itself is false (Sunday starts with the letter 'C'), it is still a boolean expression.

Instructions

1.
Determine if the following statements are boolean expressions or not. If they are, set the matching variable to the right to `"Yes"` and if not set the variable to `"No"`. Here's an example of what to do:

Example statement:

```
My dog is the cutest dog in the world.
```

This is an opinion and not a boolean expression, so you would set `example_statement` to `"No"` in the editor to the right. Okay, now it's your turn:

Statement one:

```
Dogs are mammals.
```

Statement two:

```
My dog is named Pavel.
```

Statement three:

```
Dogs make the best pets.
```

Statement four:

```
Cats are female dogs.
```

Hint
For each example, consider the following questions: *Is this objectively true or false?* If so, answer "Yes," since it's a boolean statement. If not, answer "No," since it's an opinion.

script.py

```
example_statement = "No"

statement_one = "Yes"

statement_two = "Yes"

statement_three = "No"
```

```
statement_four = "Yes"
```

Relational Operators: Equals and Not Equals
Now that we understand what boolean expressions are, let's learn to create them in Python. We can create a boolean expression by using *relational operators*.

Relational operators compare two items and return either `True` or `False`. For this reason, you will sometimes hear them called *comparators*.

The two relational operators we'll cover first are:

- Equals: `==`
- Not equals: `!=`

These operators compare two items and return `True` or `False` if they are equal or not.

We can create boolean expressions by comparing two values using these operators:

```
1 == 1     # True

2 != 4     # True

3 == 5     # False

'7' == 7   # False
```

Each of these is an example of a boolean expression.

Why is the last statement false? The `''` marks in `'7'` make it a string, which is different from the integer value `7`, so they are not equal. When using relational operators it is important to always be mindful of type.

Instructions

1.
Determine if the following boolean expressions are `True` or `False`. Input your answer as `True` or `False` in the appropriate variable to the right.

Statement one:

```
(5 * 2) - 1 == 8 + 1
```

Statement two:

```
13 - 6 != (3 * 2) + 1
```

Statement three:

```
3 * (2 - 1) == 4 - 1
```

Hint

Simplify each side of the expressions using basic order of operations, then determine if the expression is true or false. We want to use the boolean values, so make sure to use `True` and `False` rather than `"True"` and `"False"`.

Relational Operators: Equals and Not Equals

Now that we understand what boolean expressions are, let's learn to create them in Python. We can create a boolean expression by using *relational operators*.

Relational operators compare two items and return either `True` or `False`. For this reason, you will sometimes hear them called *comparators*.

The two relational operators we'll cover first are:

- Equals: `==`
- Not equals: `!=`

These operators compare two items and return `True` or `False` if they are equal or not.

We can create boolean expressions by comparing two values using these operators:

```
1 == 1     # True

2 != 4     # True

3 == 5     # False

'7' == 7   # False
```

Each of these is an example of a boolean expression.

Why is the last statement false? The `''` marks in `'7'` make it a string, which is different from the integer value `7`, so they are not equal. When using relational operators it is important to always be mindful of type.

Instructions

1.

Determine if the following boolean expressions are `True` or `False`. Input your answer as `True` or `False` in the appropriate variable to the right.

Statement one:

```
(5 * 2) - 1 == 8 + 1
```

Statement two:

```
13 - 6 != (3 * 2) + 1
```

Statement three:

```
3 * (2 - 1) == 4 - 1
```

Hint

Simplify each side of the expressions using basic order of operations, then determine if the expression is true or false. We want to use the boolean values, so make sure to use `True` and `False` rather than `"True"` and `"False"`.

script.py

```
statement_one = True

statement_two = False

statement_three = True
```

Boolean Variables

Before we go any further, let's talk a little bit about `True` and `False`. You may notice that when you type them in the code editor (with uppercase T and F), they appear in a different color than variables or strings. This is because `True` and `False` are their own special type: `bool`.

`True` and `False` are the only `bool` types, and any variable that is assigned one of these values is called a *boolean variable*.

Boolean variables can be created in several ways. The easiest way is to simply assign `True` or `False` to a variable:

```
set_to_true = True
set_to_false = False
```

You can also set a variable equal to a boolean expression.

```
bool_one = 5 != 7
bool_two = 1 + 1 != 2
bool_three = 3 * 3 == 9
```

These variables now contain boolean values, so when you reference them they will only return the `True` or `False` values of the expression they were assigned.

```
print(bool_one)    # True

print(bool_two)    # False

print(bool_three)  # True
```

Instructions

1.

Create a variable named `my_baby_bool` and set it equal to `"true"`.

Hint

It should look something like:

```
variable_name = "some string"
```

2.

Check the type of `my_baby_bool` using `type(my_baby_bool)`.

You'll have to print it to get the results to display in the terminal.

Hint

You can print the type of a variable by running:

```
print(type(variable_name))
```

3.

It's not a boolean variable! Boolean values `True` and `False` always need to be capitalized and do not have quotation marks.

Create a variable named `my_baby_bool_two` and set it equal to `True`.

Hint

Don't forget to take out the double-quotes.

4.

Check the type of `my_baby_bool_two` and make sure you successfully created a boolean variable.

You'll have to print it to get the results to display in the terminal.

Hint

You can print the type of a variable by running:

```
print(type(variable_name))
```

script.py

```
my_baby_bool = "true"

print(type(my_baby_bool))

my_baby_bool_two = True

print(type(my_baby_bool_two))
```

If Statement

"Okay okay okay, boolean variables, boolean expressions, blah blah blah, I thought I was learning how to build control flow into my code!"

You are, I promise you!

Understanding boolean variables and expressions is essential because they are the building blocks of *conditional statements*.

Recall the waking-up example from the beginning of this lesson. The decision-making process of "Is it raining? If so, bring an umbrella" is a conditional statement.

Here it is phrased in a different way:

```
If it is raining, then bring an umbrella.
```

Can you pick out the boolean expression here?

Right, `"it is raining"` is the boolean expression, and this conditional statement is checking to see if it is True.

If `"it is raining" == True` then the rest of the conditional statement will be executed and you will bring an umbrella.

This is the form of a conditional statement:

```
If [it is raining], then [bring an umbrella]
```

In Python, it looks very similar:

```
if is_raining:
  print("bring an umbrella")
```

You'll notice that instead of "then" we have a colon, `:`. That tells the computer that what's coming next is what should be executed if the condition is met.

Let's take a look at another conditional statement:

```
if 2 == 4 - 2:
  print("apple")
```

Will this code print `apple` to the terminal?

Yes, because the condition of the `if` statement, `2 == 4 - 2` is `True`.

Let's work through a couple more together.

Instructions

1.

In **script.py**, there is an `if` statement. I wrote this because my coworker Dave kept using my computer without permission and he is a real doofus. If the `user_name` is Dave, it tells him to stay off my computer.

Enter a user name in the field for `user_name` and try running the program.

Hint

On line 2, give `user_name` a string value:

```
user_name = "sonnynomnom"
```

2.

Oh no! We got a `SyntaxError`! This happens when we make a small error in the syntax of the conditional statement.

Read through the error message carefully and see if you can find the error. Then, fix it, and run the code again.

Hint

There should be an error message that looks like this:

```
 File "script.py", line 4
    if user_name = "Dave":
                 ^
SyntaxError: invalid syntax
```

Take a close look at the relational operator in the `if` statement. Is it correct?

Remember, in Python `=` is used to assign a value to a variable, where `==` is a relational operator used to see if two items are equal to each other.

Note: If you set `user_name` to something other than `"Dave"`, you'll notice that nothing is printed in the terminal.

3.

Ugh! Dave got around my security and has been logging onto my computer using our coworker Angela's user name, `angela_catlady_87`.

Set your `user_name` to be `angela_catlady_87`.

Update the program with a second `if` statement so it checks for Angela's user name as well and prints

```
"I know it is you, Dave! Go away!"
```

in response. That'll teach him!

Hint

This should have the form:

```
if user_name == "Dave":
  print("Get off my computer Dave!")

if user_name == "blah":
  print("A different message")
```

Don't forget the double equal sign `==` and the colon `:`.

script.py

```
# Enter a user name here, make sure to make it a string
user_name = "angela_catlady_87"

if user_name == "Dave":
  print("Get off my computer Dave!")

if user_name == "angela_catlady_87":
  print("I know it is you, Dave! Go away!")
```

Relational Operators II

Now that we've added conditional statements to our toolkit for building control flow, let's explore more ways to create boolean expressions. So far we know two relational operators, equals and not equals, but there are a ton (well, four) more:

- `>` greater than
- `>=` greater than or equal to
- `<` less than
- `<=` less than or equal to

Let's say we're running a movie streaming platform and we want to write a program that checks if our users are over 13 when showing them a PG-13 movie. We could write something like:

```
if age <= 13:
  print("Sorry, parental control required")
```

This function will take the user's age and compare it to the number 13. If `age` is less than or equal to 13, it will print out a message.

Let's try some more!

Instructions

1.

Create an `if` statement that checks if `x` and `y` are equal, print the string below if so:

```
"These numbers are the same"
```

Hint

Did you create an `if` statement that looks like:

```
if a == b:
  print("some message")
```

Also, make sure you copy the string response exactly.

2.

The nearby college, *Calvin Coolidge's Cool College* (or *4C*, as the locals call it) requires students to earn 120 credits to graduate.

Write an `if` statement that checks if the student has enough credits to graduate. If they do, print the string:

```
"You have enough credits to graduate!"
```

Can a student with 120 `credits` graduate from *Calvin Coolidge's Cool College*?

Hint

Make sure to use `>=` as the relational operator.

script.py

```
x = 20
y = 20

# Write the first if statement here:
if x == y:
  print("These numbers are the same")


credits = 120
```

```
# Write the second if statement here:
if credits >= 120:
  print("You have enough credits to graduate!")
```

Boolean Operators: and

Often, the conditions you want to check in your conditional statement will require more than one boolean expression to cover. In these cases, you can build larger boolean expressions using *boolean operators*. These operators (also known as *logical operators*) combine smaller boolean expressions into larger boolean expressions.

There are three boolean operators that we will cover:

- `and`
- `or`
- `not`

Let's start with `and`.

`and` combines two boolean expressions and evaluates as `True` if both its components are `True`, but `False` otherwise.

Consider the example:

```
Oranges are a fruit and carrots are a vegetable.
```

This boolean expression is comprised of two smaller expressions, `oranges are a fruit` and `carrots are a vegetable`, both of which are `True` and connected by the boolean operator `and`, so the entire expression is `True`.

Let's look at an example of some AND statements in Python:

```
(1 + 1 == 2) and (2 + 2 == 4)   # True

(1 > 9) and (5 != 6)            # False

(1 + 1 == 2) and (2 < 1)        # False

(0 == 10) and (1 + 1 == 1)      # False
```

Notice that in the second and third examples, even though part of the expression is `True`, the entire expression as a whole is `False` because the other statement is False. The fourth statement is also `False` because both components are `False`.

Instructions

1.
Set the variables `statement_one` and `statement_two` equal to the results of the following boolean expressions:

Statement one:

```
(2 + 2 + 2 >= 6) and (-1 * -1 < 0)
```

Statement two:

```
(4 * 2 <= 8) and (7 - 1 == 6)
```

2.
Let's return to *Calvin Coolidge's Cool College*. 120 credits aren't the only graduation requirement, you also need to have a GPA of 2.0 or higher.

Rewrite the `if` statement so that it checks to see if a student meets both requirements using an `and` statement.

If they do, print the string:

```
"You meet the requirements to graduate!"
```

Hint
It should look something like:

```
if credits >= 120 and [second condition]:
  print("some message")
```

Make sure to copy the string exactly!

script.py

```
statement_one = (2 + 2 + 2 >= 6) and (-1 * -1 < 0)

statement_two = (4 * 2 <= 8) and (7 - 1 == 6)

credits = 120
gpa = 3.4

if (credits >= 120) and gpa >= 2.0:
  print("You meet the requirements to graduate!")
```

Boolean Operators: or
The boolean operator `or` combines two expressions into a larger expression that is `True` if either component is `True`.

Consider the statement

```
Oranges are a fruit or apples are a vegetable.
```

This statement is composed of two expressions: `oranges are a fruit` which is `True` and `apples are a vegetable` which is `False`. Because the two expressions are connected by the `or` operator, the entire statement is `True`. Only one component needs to be `True` for an `or` statement to be `True`.

In English, `or` implies that if one component is `True`, then the other component must be `False`. This is not true in Python. If an `or` statement has two `True` components, it is also `True`.

Let's take a look at a couple of examples in Python:

```
True or (3 + 4 == 7)    # True
(1 - 1 == 0) or False   # True
(2 < 0) or True         # True
(3 == 8) or (3 > 4)     # False
```

Notice that each `or` statement that has at least one `True` component is `True`, but the final statement has two `False` components, so it is `False`.

Instructions

1.

Set the variables `statement_one` and `statement_two` equal to the results of the following boolean expressions:

Statement one:

```
(2 - 1 > 3) or (-5 * 2 == -10)
```

Statement two:

```
(9 + 5 <= 15) or (7 != 4 + 3)
```

2.

The registrar's office at *Calvin Coolidge's Cool College* has another request. They want to send out a mailer with information on the commencement ceremonies to students who have met at least one requirement for graduation (120 credits and 2.0 GPA).

Write an `if` statement that checks if a student either has 120 or more credits *or* a GPA 2.0 or higher, and if so prints:

```
"You have met at least one of the requirements."
```

Hint

It should look something like:

```
if [first condition] or [second condition]:
  print("some message")
```

Where `[first condition]` and `[second condition]` represent conditionals.

script.py

```
statement_one = (2 - 1 > 3) or (-5 * 2 == -10)

statement_two = (9 + 5 <= 15) or (7 != 4 + 3)

credits = 118
gpa = 2.0

if (credits >= 120) or (gpa >= 2.0):
  print("You have met at least one of the requirements.")
```

Boolean Operators: not

The final boolean operator we will cover is `not`. This operator is straightforward: when applied to any boolean expression it reverses the boolean value. So if we have a `True` statement and apply a `not` operator we get a `False` statement.

```
not True == False
not False == True
```

Consider the following statement:

```
Oranges are not a fruit.
```

Here, we took the `True` statement `oranges are a fruit` and added a `not` operator to make the `False` statement `oranges are not a fruit`.

This example in English is slightly different from the way it would appear in Python because in Python we add the `not` operator to the very beginning of the statement. Let's take a look at some of those:

```
not 1 + 1 == 2  # False
not 7 < 0       # True
```

Instructions

1.

Set the variables `statement_one` and `statement_two` equal to the results of the following boolean expressions:

Statement one:

```
not (4 + 5 <= 9)
```

Statement two:

```
not (8 * 2) != 20 - 4
```

2.

The registrar's office at *Calvin Coolidge's Cool College* has been so impressed with your work so far that they have another task for you.

They want you to return to a previous `if` statement and add in several checks using `and` and `not` statements:

- If a student's `credits` is not greater or equal to 120, it should print:

```
"You do not have enough credits to graduate."
```

- If their `gpa` is not greater or equal to 2.0, it should print:

```
"Your GPA is not high enough to graduate."
```

- If their `credits` is not greater than or equal to 120 *and* their `gpa` is not greater than or equal to 2.0, it should print:

```
"You do not meet either requirement to graduate!"
```

Make sure your return value matches those strings exactly. Capitalization, punctuation, and spaces matter!

Hint

The first two `if` statements should look something like:

```
if not credits >= 120:
  print("some message")
```

The third `if` statement should look something like:

```
if not (x >= 10) and not (y >= 4.0):
  print("some message")
```

script.py

```
statement_one = not (4 + 5 <= 9)

statement_two = not (8 * 2) != 20 - 4

credits = 120
gpa = 1.8
```

```
if (not credits >= 120):
  print ("You do not have enough credits to graduate.")

if (not gpa >= 2.0):
  print ("Your GPA is not high enough to graduate.")

if (not credits >= 120) and (not gpa >= 2.0):
  print ("You do not meet either requirement to graduate!")
```

Else Statements

As you can tell from your work with *Calvin Coolidge's Cool College*, once you start including lots of `if` statements in a function the code becomes a little cluttered and clunky. Luckily, there are other tools we can use to build control flow.

`else` statements allow us to elegantly describe what we want our code to do when certain conditions are **not** met.

`else` statements always appear in conjunction with `if` statements. Consider our waking-up example to see how this works:

```
if weekday:
  print("wake up at 6:30")
else:
  print("sleep in")
```

In this way, we can build if statements that execute different code if conditions are or are not met. This prevents us from needing to write `if` statements for each possible condition, we can instead write a blanket `else` statement for all the times the condition is not met.

Let's return to our `if` statement for our movie streaming platform. Previously, all it did was check if the user's age was over `13` and if so, print out a message. We can use an `else` statement to return a message in the event the user is too young to watch the movie.

```
if age >= 13:
  print("Access granted.")
else:
  print("Sorry, you must be 13 or older to watch this movie.")
```

Instructions

1.

Calvin Coolidge's Cool College has another request for you. They want you to add an additional check to a previous `if` statement. If a student is failing to meet one or both graduation requirements, they want it to print:

```
"You do not meet the requirements to graduate."
```

Add an `else` statement to the existing `if` statement.

Hint

The code should look something like:

```
if (credits >= 120) and (gpa >= 2.0):
  print("some message")
else:
  print("another message")
```

Make sure the indentations are correct!

script.py

```
credits = 120
gpa = 1.9

if (credits >= 120) and (gpa >= 2.0):
  print("You meet the requirements to graduate!")
else:
  print("You do not meet the requirements to graduate.")
```

Else If Statements

We have `if` statements, we have `else` statements, we can also have `elif` statements.

Now you may be asking yourself, what the heck is an `elif` statement? It's exactly what it sounds like, "else if". An `elif` statement checks another condition after the previous `if` statements conditions aren't met.

We can use `elif` statements to control the order we want our program to check each of our conditional statements. First, the `if` statement is checked, then each `elif` statement is checked from top to bottom, then finally the `else` code is executed if none of the previous conditions have been met.

Let's take a look at this in practice. The following `if` statement will display a "thank you" message after someone donates to a charity; there will be a curated message based on how much was donated.

```
print("Thank you for the donation!")

if donation >= 1000:
  print("You've achieved platinum status")
elif donation >= 500:
  print("You've achieved gold donor status")
elif donation >= 100:
  print("You've achieved silver donor status")
else:
  print("You've achieved bronze donor status")
```

Take a second to think about this function. What would happen if all of the `elif` statements were simply `if` statements? If you donated $1100.00, then the first three messages would all print because each `if` condition had been met.

But because we used `elif` statements, it checks each condition sequentially and only prints one message. If I donate $600.00, the code first checks if that is over 1000, which it is not, then it checks if it's over 500, which it is, so it prints that message, then because all of the other statements are `elif` and `else`, none of them get checked and no more messages get printed.

Try your hand at some other `elif` statements.

Instructions

1.

Calvin Coolidge's Cool College has noticed that students prefer to get letter grades.

Write an `if`/`elif`/`else` statement that:

- If `grade` is 90 or higher, print `"A"`
- Else if `grade` is 80 or higher, print `"B"`
- Else if `grade` is 70 or higher, print `"C"`
- Else if `grade` is 60 or higher, print `"D"`
- Else, print `"F"`

Hint

The code should look something like:

```
grade = 86

if grade >= 90:
  print("something")
elif grade >= 80:
  print("something else")
```

```
elif grade >= 70:
  print("something else")
elif grade >= 60:
  print("something else")
else:
  print("something else")
```

script.py

```
grade = 86

if grade >= 90:
  print("A")
elif grade >= 80:
  print("B")
elif grade >= 70:
  print("C")
elif grade >= 60:
  print("D")
else:
  print("F")
```

Review

Great job! We covered a ton of material in this lesson and we've increased the number of tools in our Python toolkit by several-fold. Let's review what we've learned this lesson:

- Boolean expressions are statements that can be either `True` or `False`
- A boolean variable is a variable that is set to either `True` or `False`.
- We can create boolean expressions using relational operators:
  - `==`: Equals
  - `!=`: Not equals
  - `>`: Greater than
  - `>=`: Greater than or equal to
  - `<`: Less than
  - `<=`: Less than or equal to
- `if` statements can be used to create control flow in your code.
- `else` statements can be used to execute code when the conditions of an `if` statement are not met.
- `elif` statements can be used to build additional checks into your `if` statements

Let’s put these skills to the test!

Instructions

Optional: Little Codey is an interplanetary space boxer, who is trying to win championship belts for various weight categories on other planets within the solar system.

Write a **space.py** program that helps Codey keep track of their target weight by:

1. Checks which number `planet` is equal to.
2. It should then compute their weight on the destination planet.

Here is the table of conversions:

#	Planet	Relative Gravity
1	Venus	0.91
2	Mars	0.38
3	Jupiter	2.34
4	Saturn	1.06
5	Uranus	0.92
6	Neptune	1.19

To compute their weight on the planet they are fighting on, multiply their earth weight and the relative gravity of that planet.

```
  if planet == 1:
    weight = weight * 0.91
  elif planet == 2:
    weight = weight * 0.38
  elif planet == 3:
    weight = weight * 2.34
  elif planet == 4:
    weight = weight * 1.06
  elif planet == 5:
    weight = weight * 0.92
  elif planet == 6:
    weight = weight * 1.19

  print("Your weight:", weight)
```

Full solution code can be found here.

space.py

```
print("   4. Saturn  5. Uranus  6. Neptune\n")

weight = 185
planet = 3

# Write an if statement below:
if planet == 1:
  weight = weight * 0.91
elif planet == 2:
  weight = weight * 0.38
elif planet == 3:
  weight = weight * 2.34
elif planet == 4:
  weight = weight * 1.06
elif planet == 5:
  weight = weight * 0.92
elif planet == 6:
  weight = weight * 1.19

print("Your weight:", weight)
```