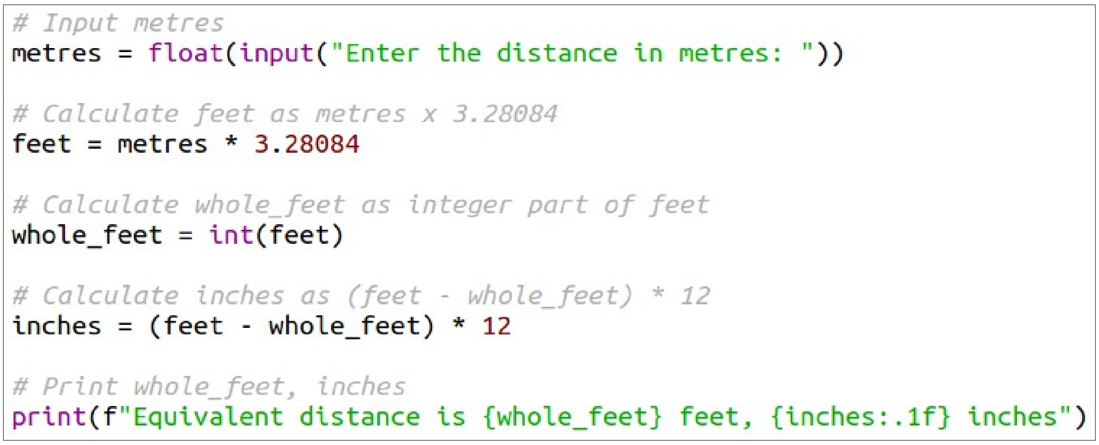
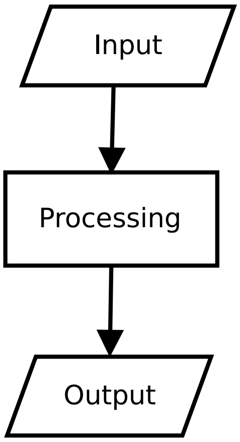
Table of Contents

Introduction to Control Structures

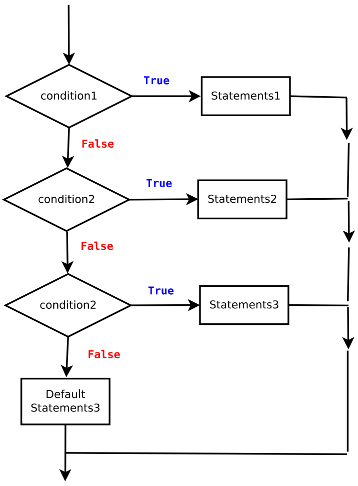
The previous programs all followed the same format, involving a sequence of statements, executed one after another, from top to bottom. For example:



```
# Input metres
metres = float(input("Enter the distance in metres: "))

# Calculate feet as metres x 3.28084
feet = metres * 3.28084

# Calculate whole_feet as integer part of feet
whole_feet = int(feet)

# Calculate inches as (feet - whole_feet) * 12
inches = (feet - whole_feet) * 12

# Print whole_feet, inches
print(f"Equivalent distance is {whole_feet} feet, {inches:.1f} inches")
```



Control structures provide a mechanism to control the flow of statement execution. Python provides 3 control structures:

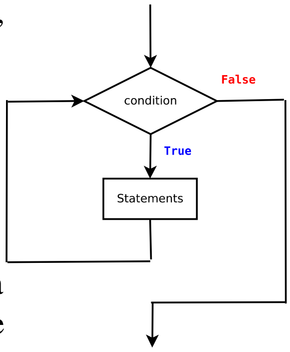


Decision Making with if-elif-else

This is a *decision making* structure, also called *choice* or *selection*.

```
if condition1:
    statements1
elif condition2:
    statements2
elif condition3:
    statements3
else:
    default statements
```

The program chooses between two or more alternatives (statement blocks), depending on the evaluation of one or more *boolean conditions*.

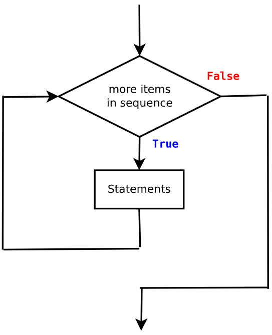


while Repetition Loop

This is a *repetition* structure, also called *iteration*.

```
while condition:
    statement(s)
```

The program repeats a block of statements while a boolean condition evaluates as `True`. That is, the loops repeats as long as the boolean condition is `True`.



for Iteration Loop

This is another *repetition* structure, but instead of using a condition, it *iterates* through the values in a sequence, one at a time.

```
for variable in sequence:
    statement(s)
```

The program repeats a block of statements for each value in the sequence provided.

Control Structures in Python 1: Conditions and Decision Making with if-elif-else

Conditions

Conditions, also caled Conditional Expressions, or Boolean Expressions, are used in decision making with `if-elif-else` structures and with `while` repetition loops.

When evaluated, its result is either:

`True` meaning the condition holds
or **`False`** meaning the condition does not hold.

Conditions are effectively questions that have a yes/no answer. “Yes” is represented in Python as `True` and “No” is represented in Python as `False`

Conditions using Relational Operators

Conditions involving simple comparisons have the following form:

	value 1	relational operator	value 2
e.g.	temperature	>	27

A value can be:

- a simple value, such as 0, “a”, 38.6, “Hello World”
- the value of a variable, such as `temperature`
- an *expression* (combination of values, operators and variables), which must be evaluated, e.g. `9.0/5 * celsius + 32`

Relational operators are used to compare two values with each other:

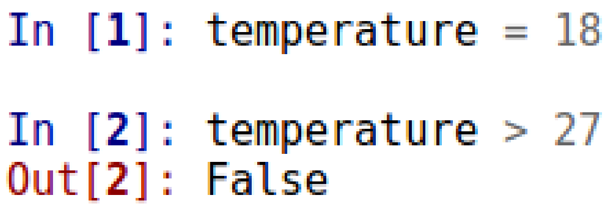
Relational Operators

Operator	Meaning	Example
<	Less Than	3 < 5 evaluates as `True` 5 < 3 evaluates as `False`
>	Greater Than	5 > 3 evaluates as `True`. 3 > 5 evaluates as `False`
<=	Less Than or Equal To	x = 3; y = 6; x <= y evaluates as `True`.
>=	Greater Than or Equal To	x = 4; y = 3; x >= 3 evaluates as `True`.
==	Equal To	x = 2; y = 2; x == y evaluates as `True`.
!=	Not Equal To	x = 2; y = 3; x != y evaluates as `True`.

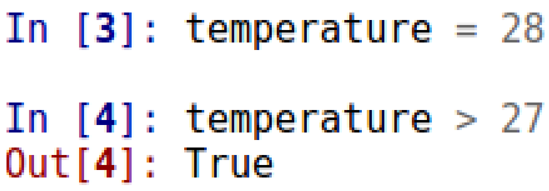
Examples

The following examples use the iPython interactive interpreter to demonstrate conditions using relational operators. `In` means information I have typed, such as statements of code. `Out` means the output from the code statement. The number in the square brackets indicates the order of the items, so `[1]` was the first item, `[2]` was the second, and so on.

Temperature above 27 degrees?

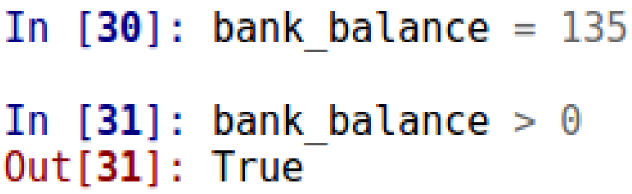


```
In [1]: temperature = 18

In [2]: temperature > 27
Out[2]: False
```

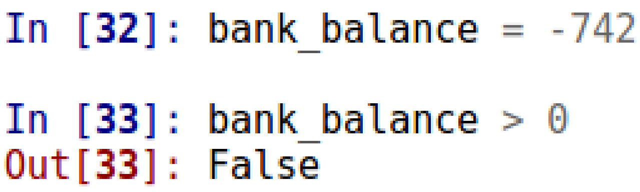


```
In [3]: temperature = 28

In [4]: temperature > 27
Out[4]: True
```

Bank Account in Credit?

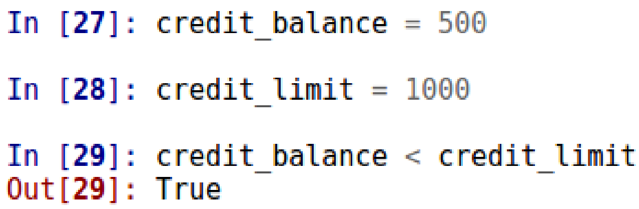


```
In [30]: bank_balance = 135

In [31]: bank_balance > 0
Out[31]: True
```

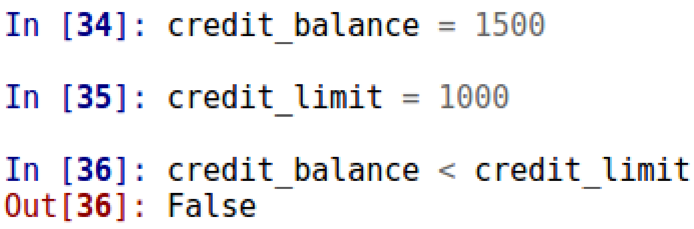


```
In [32]: bank_balance = -742

In [33]: bank_balance > 0
Out[33]: False
```

Money to spend on Credit Card?

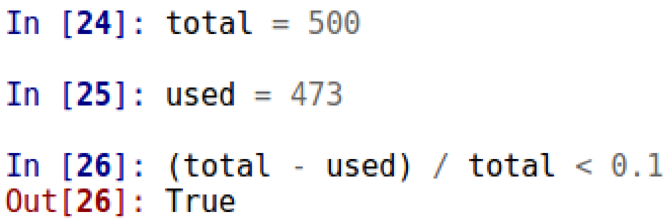


```
In [27]: credit_balance = 500

In [28]: credit_limit = 1000

In [29]: credit_balance < credit_limit
Out[29]: True
```

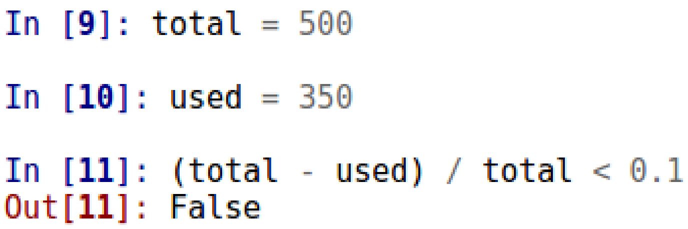


```
In [34]: credit_balance = 1500

In [35]: credit_limit = 1000

In [36]: credit_balance < credit_limit
Out[36]: False
```

Free space less than 10% of total space?
This example shows the use of an expression to be calculated in a conditon:



```
In [24]: total = 500

In [25]: used = 473

In [26]: (total - used) / total < 0.1
Out[26]: True
```

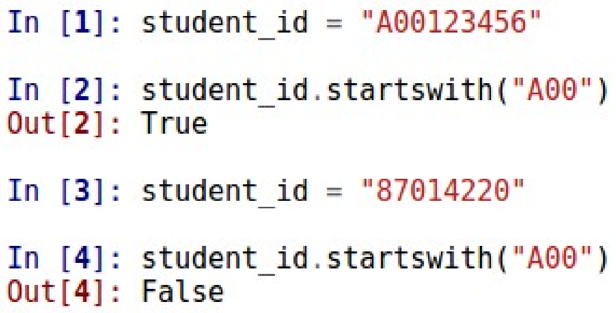


```
In [9]: total = 500

In [10]: used = 350

In [11]: (total - used) / total < 0.1
Out[11]: False
```

Boolean Functions as Conditions

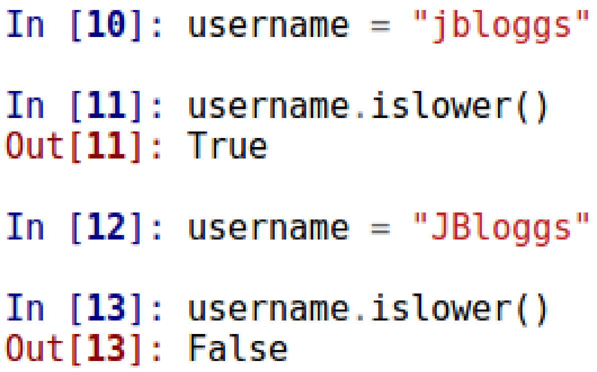
Boolean functions/methods return `True` or `False`. They can be used in, or as, conditional expressions. For example, a number of String methods and are used to check specific features of a string:

Does the Student ID start with “`A00`”?



```
In [1]: student_id = "A00123456"

In [2]: student_id.startswith("A00")
Out[2]: True

In [3]: student_id = "87014220"

In [4]: student_id.startswith("A00")
Out[4]: False
```

Are the letters in the username all lowercase? (This also checks for at least 1 letter in the string; if the string does not contain at least 1 lowercase letter, the method returns `False`).



```
In [10]: username = "jbloggs"

In [11]: username.islower()
Out[11]: True

In [12]: username = "JBloggs"

In [13]: username.islower()
Out[13]: False
```

Boolean Operators `and` and `or`

The Boolean operators **`and`** and **`or`** are used to combine two Boolean conditions and produce a Boolean result, `True` or `False`.

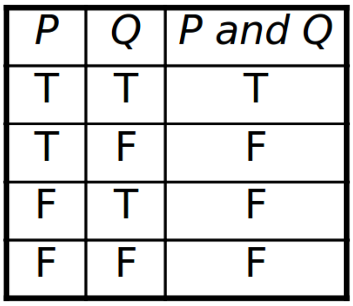
`condition1 and condition2`

The combined condition is true exactly when ***both*** of the expressions are true.

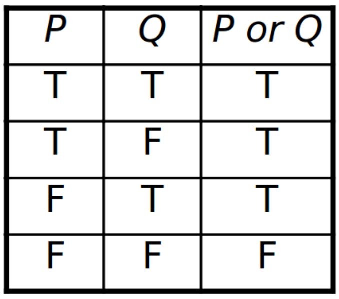
`condition1 or condition2`

The combined condition is True when ***either*** expression is true (or both). The only time **`or`** evaluates as `False` is when both conditions are `False`.

This information is often summarised in a “Truth Table”, where P and Q represent simpler Boolean expressions.



P	Q	P and Q
T	T	T
T	F	F
F	T	F
F	F	F



P	Q	P or Q
T	T	T
T	F	T
F	T	T
F	F	F

Example: `and`

A program is to process a student's mark out of a hundred; the logic to check for a valid mark is:

greater than or equal to 0 AND less than or equal to 100

The boolean expression is:

```
mark >= 0 and mark <= 100
```

In this example, the combined condition is true if and only if both simple conditions are true:

mark	mark >= 0	mark <= 100	valid mark?
75	True	True	True
110	True	False	False
-25	False	True	False

Note:

Unlike most programming languages, Python allows the following syntax:

```
0 <= mark <= 100
```

Other Examples

No money in the bank and credit card maxed out?

```
bank_balance <= 0 and credit >= limit
```

You can combine any number of **and**s, for example to check for healthy cholesterol levels:

```
total <= 5 and ldl <= 3 and hdl > 1
```

Examples: **or**

Some amusement parks have age and height restrictions for their rides,

```
age < 3 or height < 1.02
```

If you're under 14 or more than 235kg, you can't bungee jump:

```
age < 14 or weight > 235
```

Combining Multiple Conditions with and and or

We can make arbitrarily complex Boolean conditions by combined multiple conditions. However, if the combined condition contains a mixture of the `and` and `or` operators, the evaluation of the combined condition relies on the ***precedence rules*** for the operators, i.e. the order in which they appear.

For example, suppose we have a program to identify mature or part-time engineering students. There are three conditions to be combined:

```
    age > 23
    status == “part-time”
and school = “Engineering”
```

Is there a difference between the following two ways of combining the conditions? Does the order matter?

```
age > 23 or status == “part-time” and school = “Engineering”
```

and

```
school = 'engineering' and age > 23 or status == 'part-time'
```

Let’s consider some values, and examine the overall result in each case.

```
Version 1   age > 23 or status == “part-time” and school = “Engineering”
Version 2   school = “Engineering” and age > 23 or status == “part-time”
```

age	status	school	Version 1	Version 2
25	part-time	Engineering	True	True
25	full-time	Engineering	True	True
18	part-time	Engineering	True	True
18	full-time	Engineering	False	False
18	part-time	Science	False	True
25	full-time	Science	True	False

The two combined conditions are evaluated differently. So the order does matter.

The rule is: **and** is evaluated before **or**, unless brackets are used to override this

So the expression `age > 23 or status == “part-time” and school = “Engineering”`
is equivalent to `age > 23 or (status == “part-time” and school = “Engineering”)`

and the expression `school = “Engineering” and age > 23 or status == 'part-time'`
is equivalent to `(school = “Engineering” and age > 23) or status == 'part-time'`

In this case, to check if some or is a mature or part-time Engineering student, what's required is:

```
    (age > 23 or status == 'part-time') and school = “Engineering”
```

or equivalently

```
    school = “Engineering” and (age > 23 or status == “part-time”)
```

If unsure, use brackets in a combined condition involving **and** and **or** operators.

The `not` operator

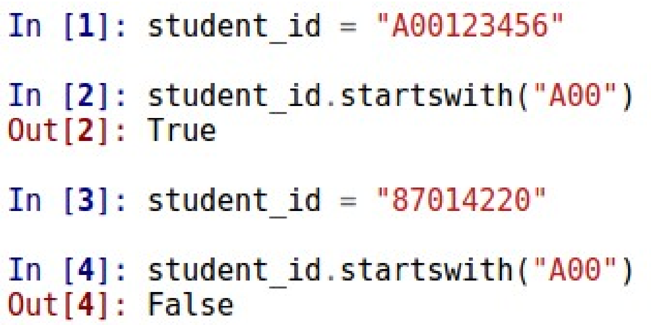
The `not` operator yields the *opposite* of a Boolean expression. It reverses the "truth" of a condition. The truth table is:

P	not P
T	F
F	T

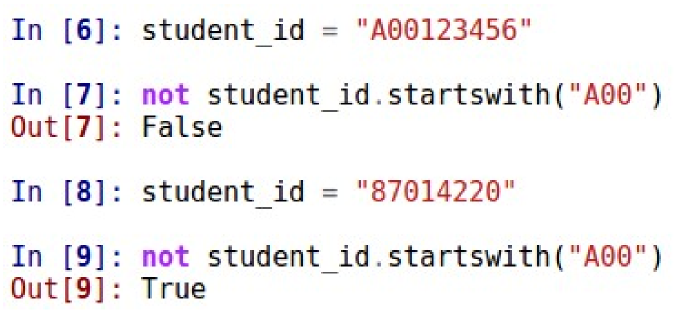
If a Boolean Expression P is **True** then **not** P is **False**.
If a Boolean Expression P is **False** then **not** P is **True**.

Examples:
This is particularly handy when you want to check if a string method evaluates as False.

For example, check if the Student ID does not start with the "A00"



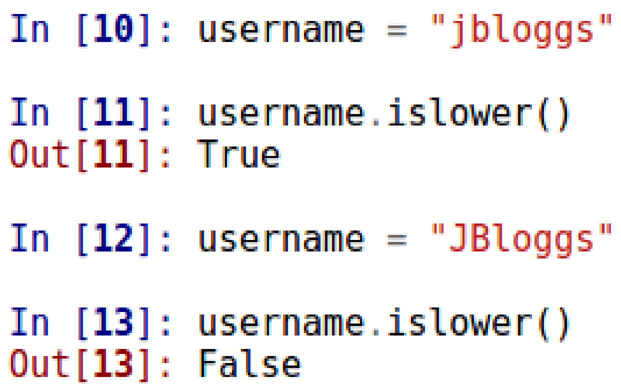
```
In [1]: student_id = "A00123456"

In [2]: student_id.startswith("A00")
Out[2]: True

In [3]: student_id = "87014220"

In [4]: student_id.startswith("A00")
Out[4]: False
```



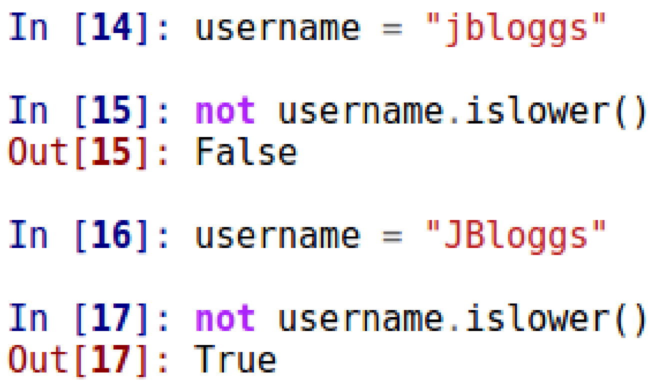
```
In [6]: student_id = "A00123456"

In [7]: not student_id.startswith("A00")
Out[7]: False

In [8]: student_id = "87014220"

In [9]: not student_id.startswith("A00")
Out[9]: True
```

This is useful for input validation, e.g. ensuring a valid AIT student ID has been input.

The following example checks if the letters in a username are not all lowercase:

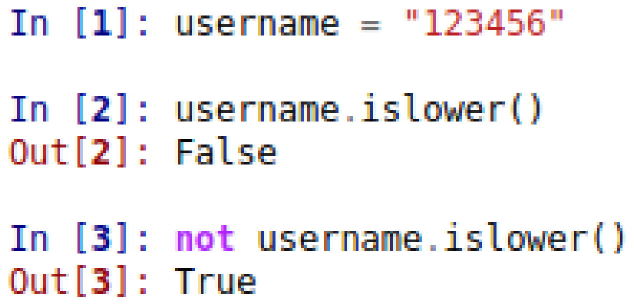


```
In [10]: username = "jbloggs"

In [11]: username.islower()
Out[11]: True

In [12]: username = "JBloggs"

In [13]: username.islower()
Out[13]: False
```



```
In [14]: username = "jbloggs"

In [15]: not username.islower()
Out[15]: False

In [16]: username = "JBloggs"

In [17]: not username.islower()
Out[17]: True
```

Note that, the `islower()` method also returns false if there are no lowercase letters in the string:



```
In [1]: username = "123456"

In [2]: username.islower()
Out[2]: False

In [3]: not username.islower()
Out[3]: True
```

More Falseness

Python lets you use any value where it expects a Boolean, for example with `if-elif` and `while`.

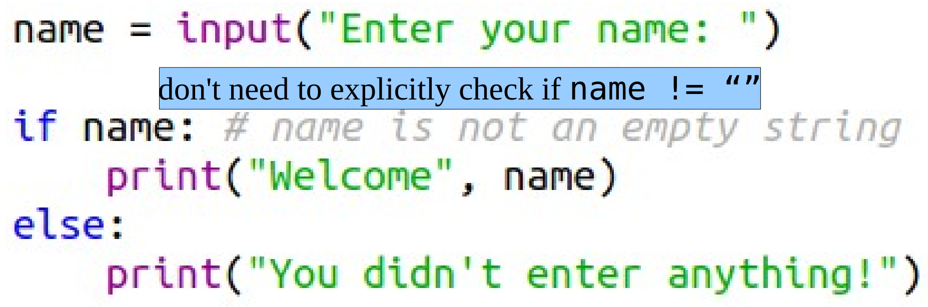
The following all evaluate as `False`:

`None`
an empty list `[]` tuple `()` or dictionary `{}`
an empty string `""`
The number zero: 0 or 0.0

Everything else evaluates as `True`.

This makes it easy to check for null values (`None`), empty strings, or empty data structures.

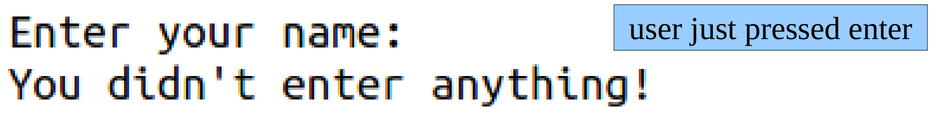
For example, you could check if a user has not provided any input in response to a prompt:



```
name = input("Enter your name: ")

if name: # name is not an empty string
    print("Welcome", name)
else:
    print("You didn't enter anything!")
```

don't need to explicitly check if `name != “”`

Sample Output



```
Enter your name:
You didn't enter anything!
```

user just pressed enter

Decision Making with `if-elif-else`

We need to be able to change the flow of a program to suit a particular situation. In particular, computer programs need to “decide” what to do based on different situations.

Examples of Decisions

Program	**Decision**
ATM	Has a card been entered?
	Does the PIN match the card?
Windows	Has the user logged in correctly?
Spyder	Has the user clicked on Save?
Fridge	Is the door open more than 3 seconds?

Decision making structures allow a program to execute different instructions for different cases. This allows the program to “choose” an appropriate course of action, depending on the situation.

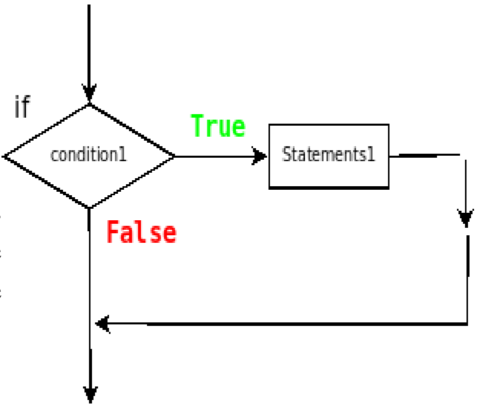
Python's only decision making structure is **`if-elif-else`** which chooses between two or more alternatives (statement blocks), depending on the evaluation of one or more *boolean conditions*.

One-Way Decisions with `if`

Python uses `if` statements to implement decisions. A ***One-Way Decision*** is required when the program needs to take an action, i.e. execute one or more statements, when a condition evaluates as `True`, and otherwise the program takes no action.

The syntax is

```
if condition:
     statement(s)
```



`statement(s)` are a sequence of one or more statements indented under, and associated with, the `if` statement. The *`condition`* is a check to see if these statements should be executed.

<u>*Important:*</u>
Python uses indentation (“whitespace”: spaces or tabs) to signify blocks of code, rather than braces `{}`.

The line containing the `if` must end with a colon **:** indicating that an indented block is to follow.

```
if condition:
     block
```

signifies the start of an indented block

De-denting (unindenting) signifies the end of the block.

https://unspecified.wordpress.com/2011/10/18/why-pythons-whitespace-rule-is-right/

Example

A program is required which

- inputs a temperature value (in degrees Celsius)
- then displays a “Status Yellow Warning” message if the temperature is above 27

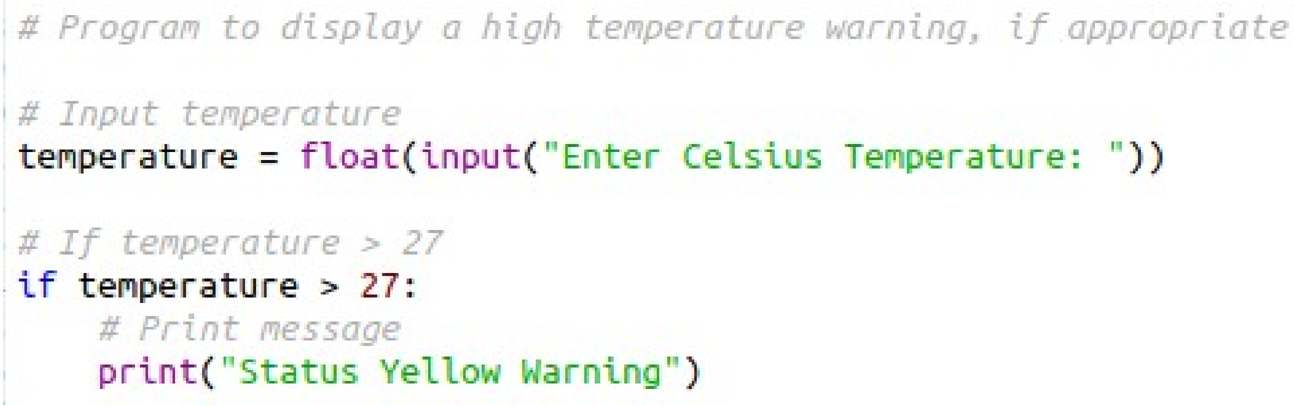
Sample Values

Input: temperature	Output
18	
27	
28	“Status Yellow Warning”

Specification Table

Input	Processing	Output
temperature	Input temperature If temperature > 27 Print message	message

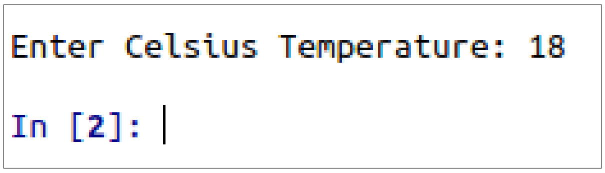
Python Program



```python
# Program to display a high temperature warning, if appropriate

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# If temperature > 27
if temperature > 27:
    # Print message
    print("Status Yellow Warning")
```

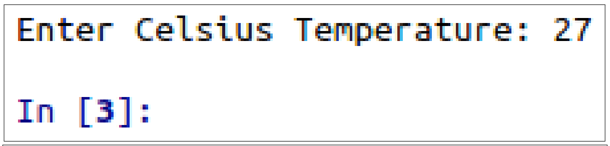
Sample Output

Here is the output from executing the program 3 separate times.



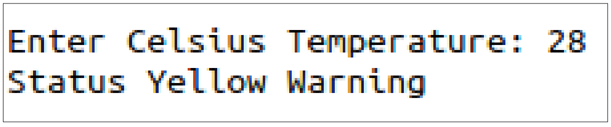
```
Enter Celsius Temperature: 18

In [2]: |
```

No output



```
Enter Celsius Temperature: 27

In [3]:
```

No output



```
Enter Celsius Temperature: 28
Status Yellow Warning
```

Output

Testing

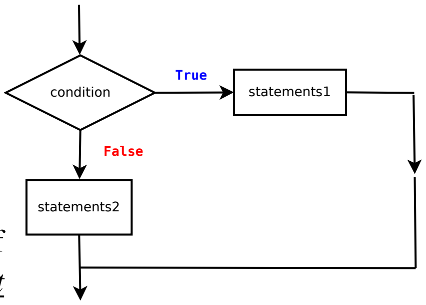
Input	Output		Pass Y/N?
	Expected	Actual	
18	(no output)	(no output)	**Y**
27	(no output)	(no output)	**Y**
28	Status Yellow Warning	Status Yellow Warning	**Y**

Two-Way Decisions with `if-else`

A ***Two-Way Decision*** involves a situation where the program needs to perform one action (i.e. execute one or more statements) if a condition evaluates as `True`, otherwise the program will perform a different action (a separate block of statements).

It is represented using `if-else`, and is implemented by attaching an `else` clause onto an `if` clause.

```
if condition:
    statements1
else:
    statements1
```



This is called an `if-else` structure. Notice the colon **:** at the end of the `else` line. As with the colon at the end of the `if` line, this *must* be included as it indicates an associated block of code, which must be indented.

For example, the previous program can be extended to display a "Status Green" message if the temperature is less than or equal to 27 degrees Celsius.

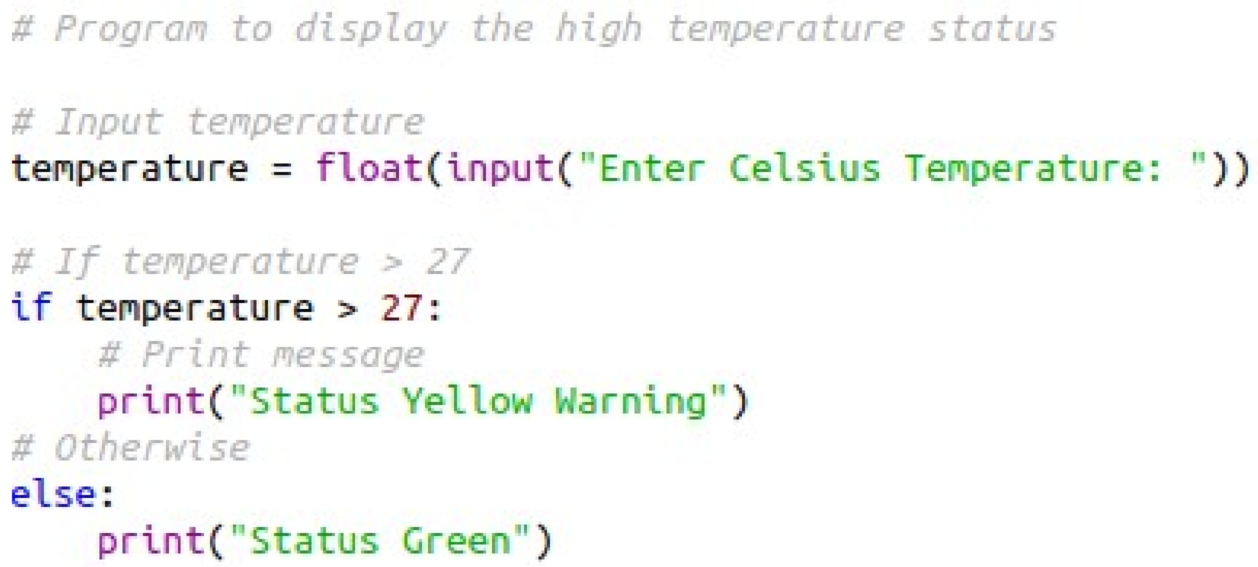
Sample Values

Input: temperature	Output
18	"Status Green"
27	"Status Green"
28	"Status Yellow Warning"

Specification Table

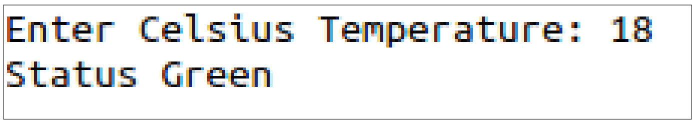
Input	Processing	Output
temperature	Input temperature If temperature > 27 Print Status Yellow message Else Print Status Green message	message

Python Program

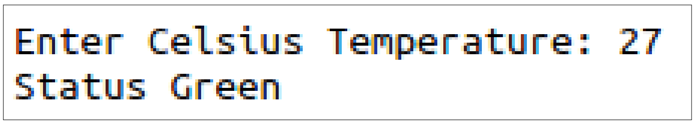


```
# Program to display the high temperature status

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# If temperature > 27
if temperature > 27:
    # Print message
    print("Status Yellow Warning")
# Otherwise
else:
    print("Status Green")
```

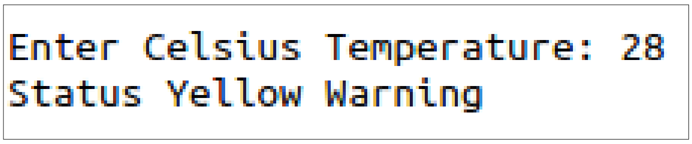
Sample Output



```
Enter Celsius Temperature: 18
Status Green
```



```
Enter Celsius Temperature: 27
Status Green
```



```
Enter Celsius Temperature: 28
Status Yellow Warning
```

Testing

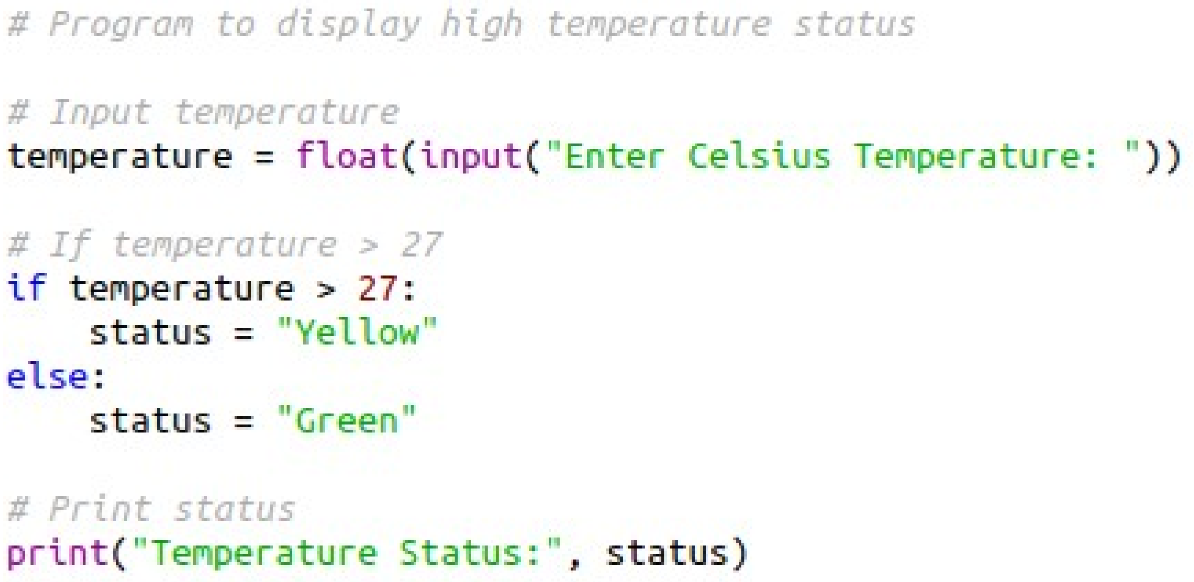
Input	**Output**		**Pass Y/N?**
	Expected	**Actual**	
18	Status Green	Status Green	**Y**
27	Status Green	Status Green	**Y**
28	Status Yellow Warning	Status Yellow Warning	**Y**

Alternative Version

This is a slightly different version, which involves using a variable to store the status message. The `if-else` structure is used to determine the message to be stored; after the `if-else` structure, the message will be displayed, by displaying the contents of the variable.

Input	Processing	Output
temperature	Input temperature If temperature > 27 Set status to Yellow Else Set status to Green Print status	status

Python Program



```
# Program to display high temperature status

# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# If temperature > 27
if temperature > 27:
    status = "Yellow"
else:
    status = "Green"

# Print status
print("Temperature Status:", status)
```

The output is the same as for the previous version.

Python's Ternary Conditional Operator

A simple `if-else` can be expressed in one line using Python's Ternary Conditional Operator. The syntax is:

```
value1 if condition else value2
```

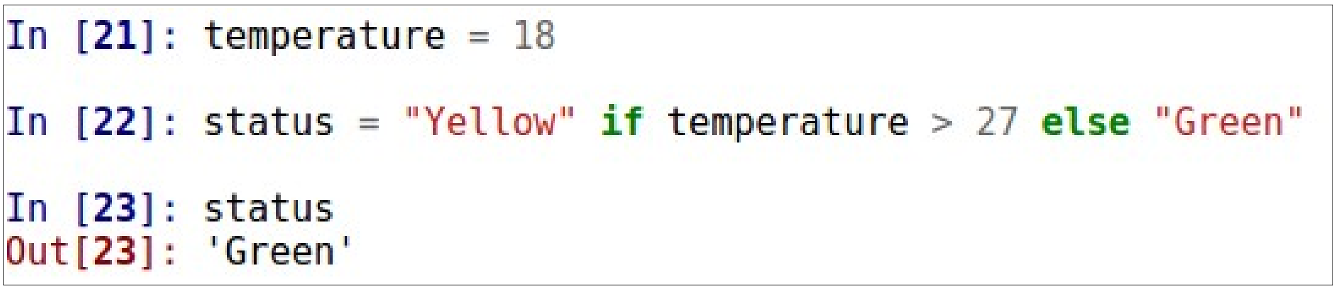
Explanation:

If the *condition* is **True**,
value1 is used
otherwise
value2 is used

For Example:

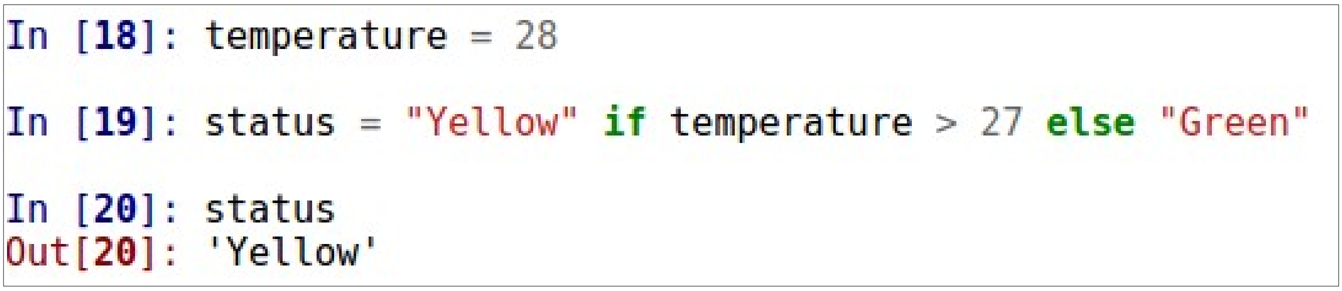
```
status = “Yellow” if temperature > 27 else “Green”
```

Here's how it appears in the iPython interactive interpreter, first with a temperature value of 18 (and so the `else` block is activated):



```
In [21]: temperature = 18

In [22]: status = "Yellow" if temperature > 27 else "Green"

In [23]: status
Out[23]: 'Green'
```

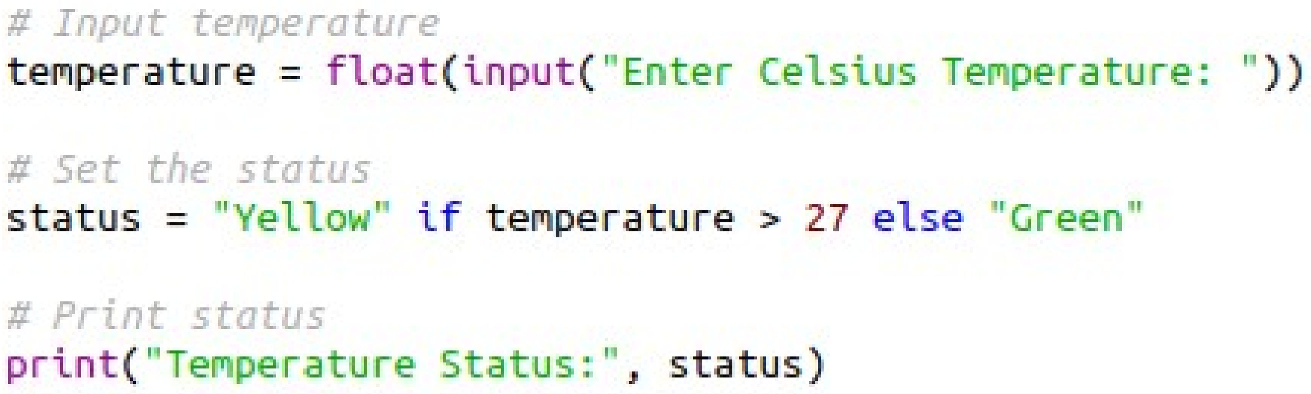
Here it is with a temperature of 28, and so the code associated with the if is executed.



```
In [18]: temperature = 28

In [19]: status = "Yellow" if temperature > 27 else "Green"

In [20]: status
Out[20]: 'Yellow'
```

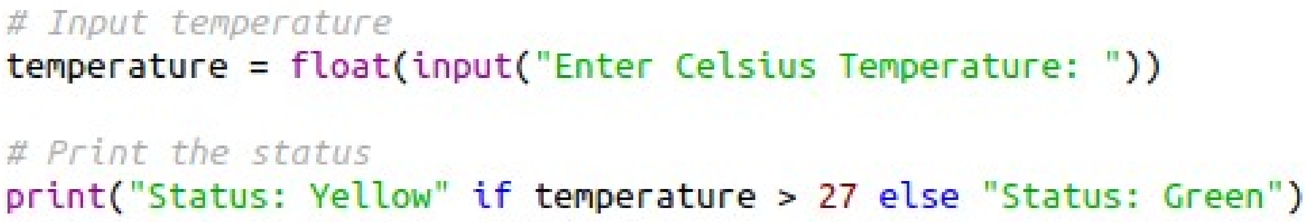
Python Program

This leads to a slightly shorter program:



```
# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# Set the status
status = "Yellow" if temperature > 27 else "Green"

# Print status
print("Temperature Status:", status)
```

You can use this in a **print** function, which makes the program even shorter:



```python
# Input temperature
temperature = float(input("Enter Celsius Temperature: "))

# Print the status
print("Status: Yellow" if temperature > 27 else "Status: Green")
```

The **print** function prints “Status Yellow” if the temperature is greater than 27, otherwise it prints “Status Green”.

Points to consider:

- There are fewer lines of code
- There is no difference in performance
- The order is different from corresponding structure in traditional programming languages
  `<condition> ? <expression1> : <expression2>`
- Debugging the classic `if-else` is easier
- The syntax could be misleading (precedence rules)

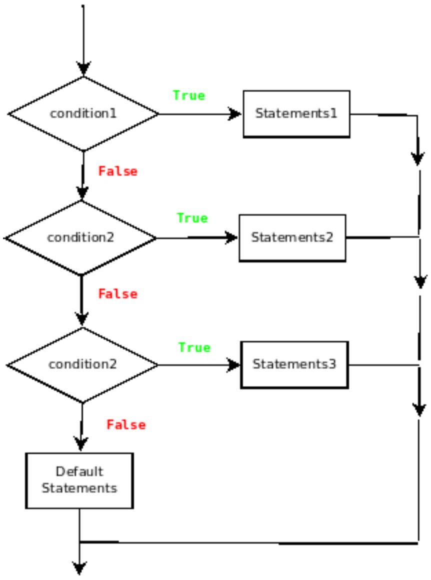
https://blog.softhints.com/python-3-if-else-one-line-or-ternary-operator/

Multi-way Decisions with `if-elif-else`

A *Multi-Way Decision* involves more than two possible alternatives. The program needs to perform one action (i.e. execute one or more statements) if the first condition evaluates as `True`, otherwise the program will perform a different action (a separate block of statements), if the second condition (or some further condition) evaluates as `True`, otherwise the program will take some specified default action.

It is represented using `if-elif-else:`

```
if condition1:
    statements1
elif condition2:
    statements2
elif condition3:
    statements3
else:
    default statements
```



The keyword `elif` represents “Else If”.

Example

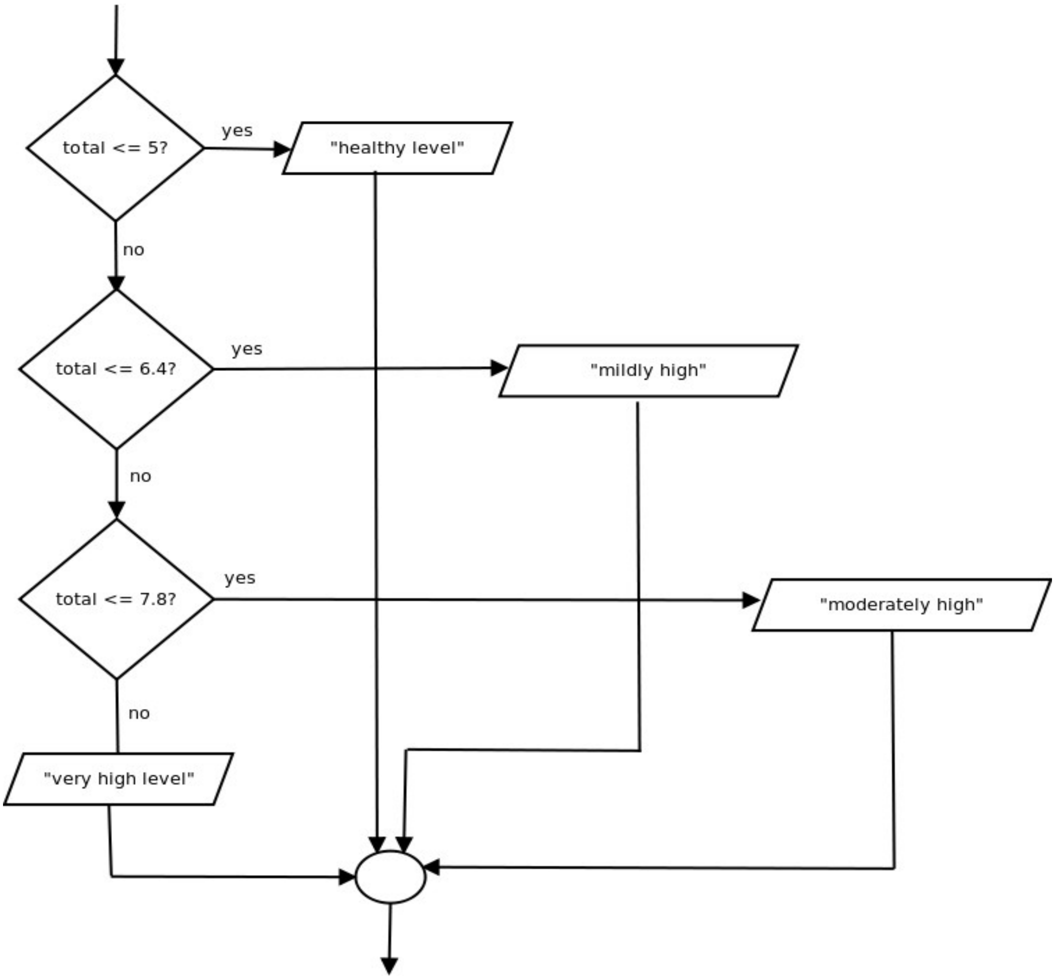
High Cholesterol levels may lead to heart disease. There are two types of Cholesterol:

- HDL (high density lipoprotein) “good” cholesterol
- LDL (low density lipoprotein) “bad” cholesterol

A diagnosis of high cholesterol is based on calculating the total (HDL+LDL) cholesterol:

Total Cholesterol	Description
0 up to 5	Healthy cholesterol level
Above 5 and up to 6.4	Mildly high cholesterol level
Above 6.4 and up to 7.8	Moderately high level
Above 7.8	Very high cholesterol

This requires a ***multi-way decision*** structure.

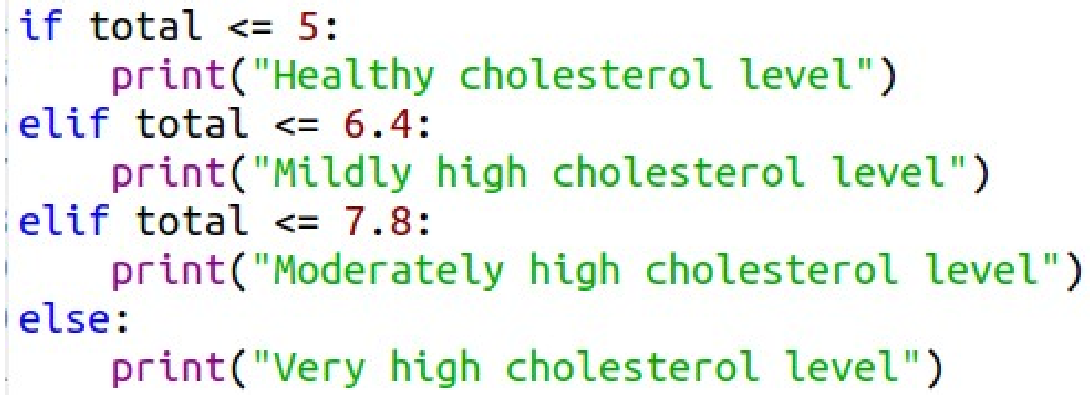


The logic is as follows:

- If the total is 5 or less, then display “healthy cholesterol”, and it's finished.
- Otherwise (so the total is not 5 or less, i.e. greater than 5), if the total is 6.4 or less (and greater than 5), then display “mildly high level”
- Otherwise (so the total is not 6.4 or less, i.e. more than 6.4), if the total is 7.8 or less (and greater than 6.4), then display “moderately high level”.
- Otherwise (so the total is not 7.8 or less, i.e. more than 7.8), display “very high level”.

Python Implementation

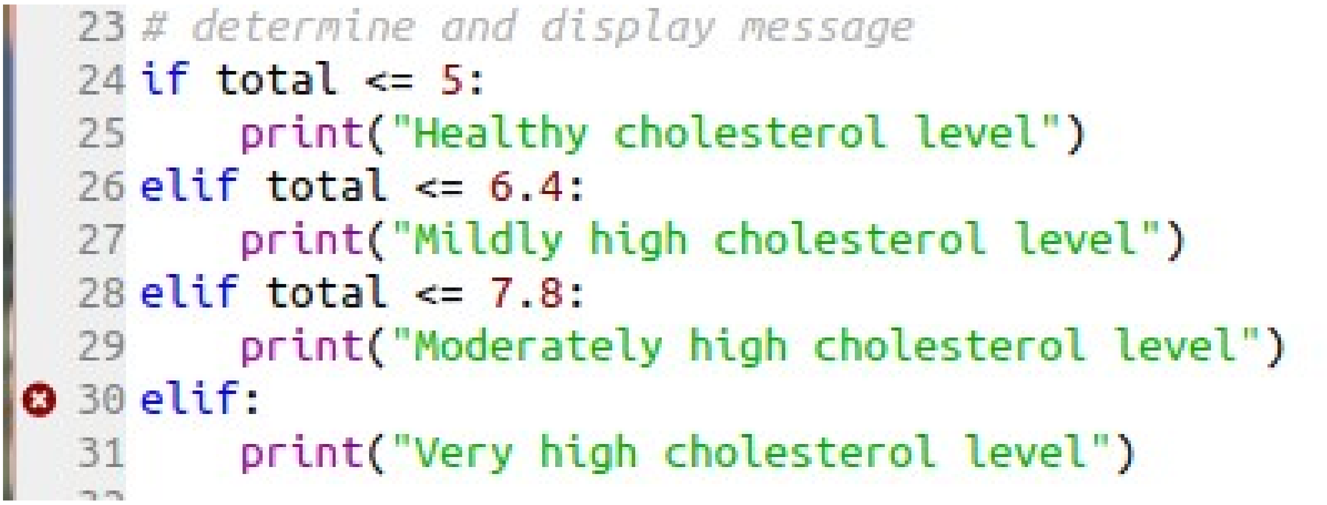
The decision structure to determine the appropriate message is:



```
if total <= 5:
    print("Healthy cholesterol level")
elif total <= 6.4:
    print("Mildly high cholesterol level")
elif total <= 7.8:
    print("Moderately high cholesterol level")
else:
    print("Very high cholesterol level")
```

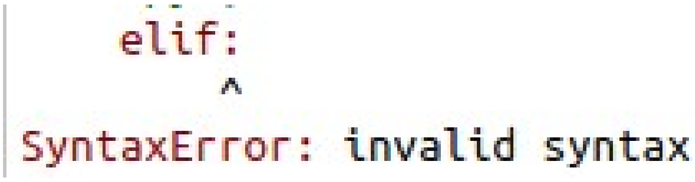
Watchout! Common Mistakes

1.The `elif` must have a condition associated with it.



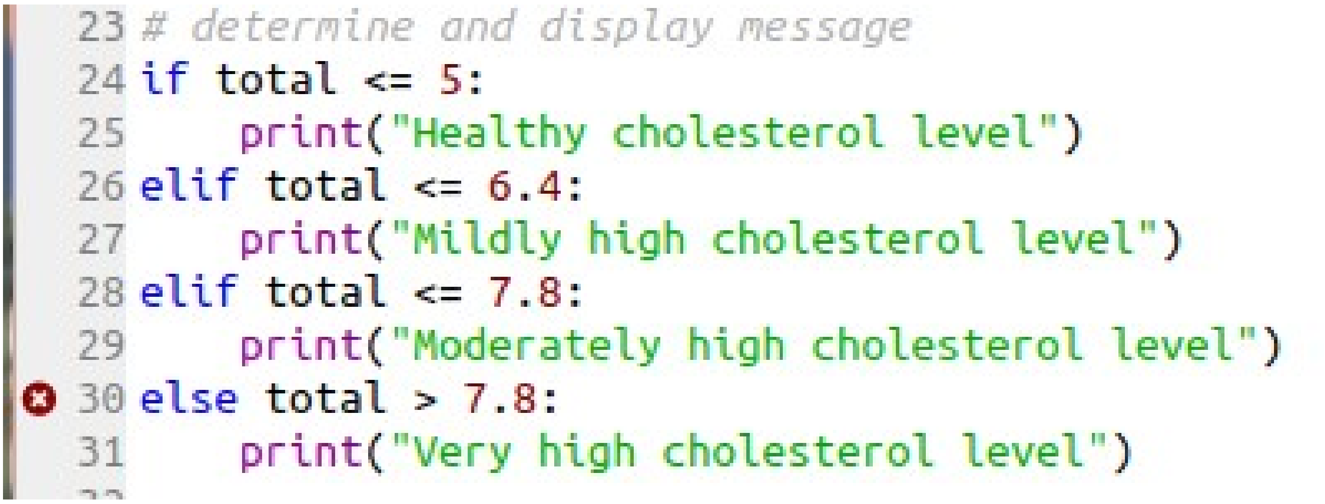
```
# determine and display message
if total <= 5:
    print("Healthy cholesterol level")
elif total <= 6.4:
    print("Mildly high cholesterol level")
elif total <= 7.8:
    print("Moderately high cholesterol level")
elif:
    print("Very high cholesterol level")
```

This causes an error:



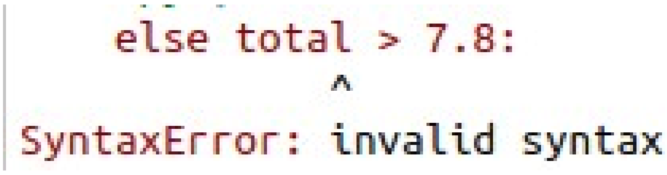
```
    elif:
        ^
SyntaxError: invalid syntax
```

2.A more common mistake is to include a condition with the `else`:



```
# determine and display message
if total <= 5:
    print("Healthy cholesterol level")
elif total <= 6.4:
    print("Mildly high cholesterol level")
elif total <= 7.8:
    print("Moderately high cholesterol level")
else total > 7.8:
    print("Very high cholesterol level")
```

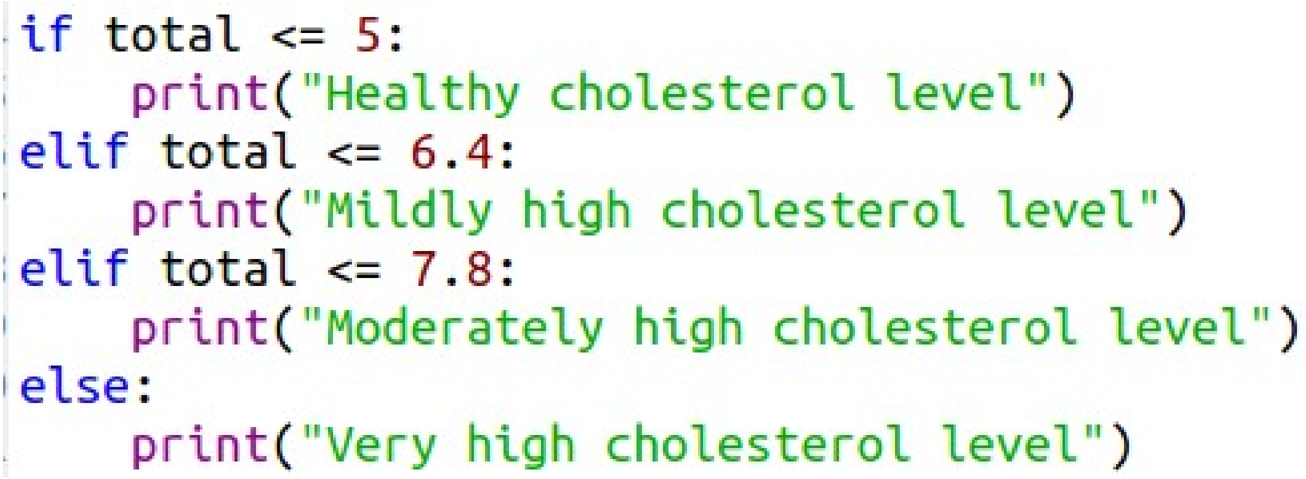
This also causes an error:



```
    else total > 7.8:
             ^
SyntaxError: invalid syntax
```

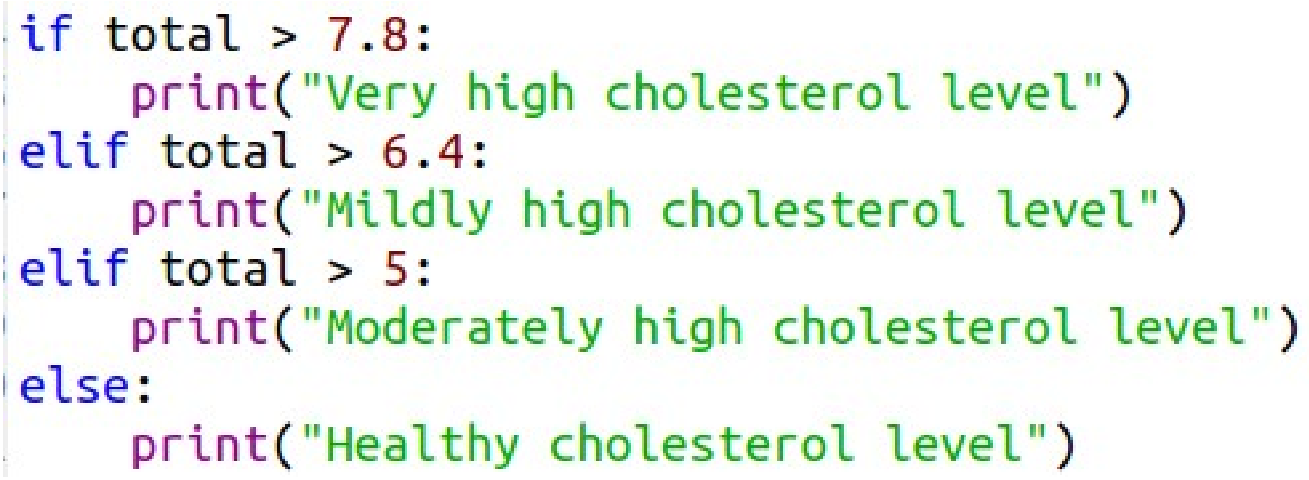
Processing Values in Order

Notice how the `if-elif-else` structure processed the cholesterol values in order, lowest to highest:



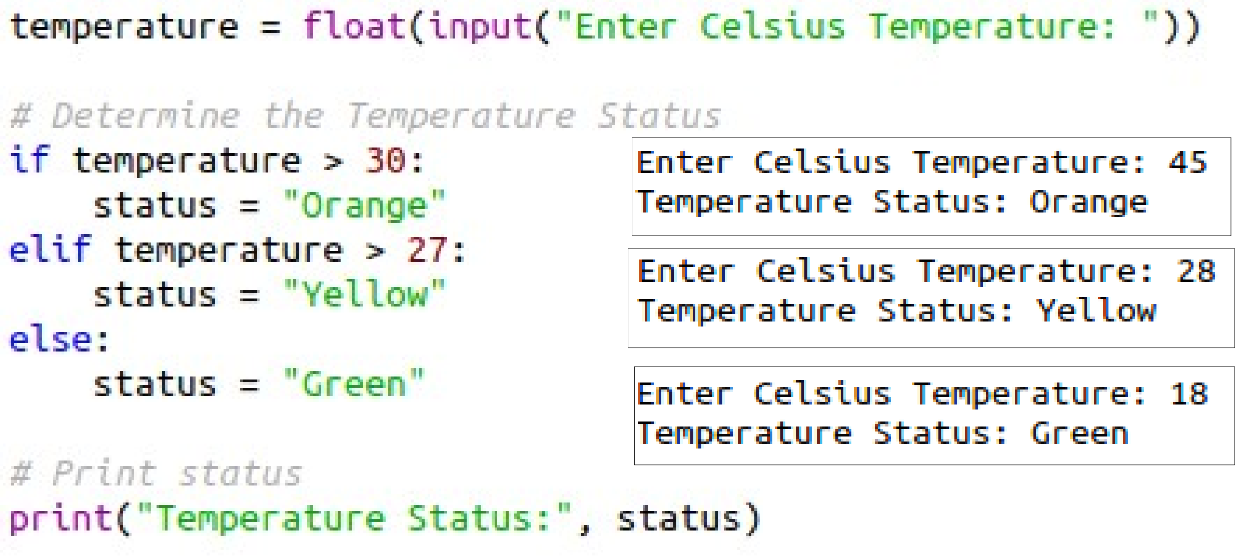
```
if total <= 5:
    print("Healthy cholesterol level")
elif total <= 6.4:
    print("Mildly high cholesterol level")
elif total <= 7.8:
    print("Moderately high cholesterol level")
else:
    print("Very high cholesterol level")
```

Alternatively, you could process the values in order, highest to lowest:



```
if total > 7.8:
    print("Very high cholesterol level")
elif total > 6.4:
    print("Mildly high cholesterol level")
elif total > 5:
    print("Moderately high cholesterol level")
else:
    print("Healthy cholesterol level")
```

In another example, the following program determines temperature warnings using `if-elif-else` to process the values in order:



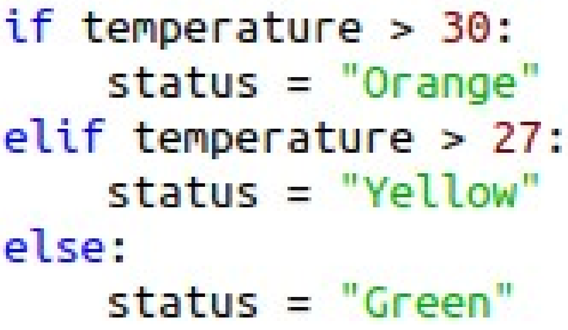
```
temperature = float(input("Enter Celsius Temperature: "))

# Determine the Temperature Status
if temperature > 30:
    status = "Orange"
elif temperature > 27:
    status = "Yellow"
else:
    status = "Green"

# Print status
print("Temperature Status:", status)
```

```
Enter Celsius Temperature: 45
Temperature Status: Orange
```

```
Enter Celsius Temperature: 28
Temperature Status: Yellow
```

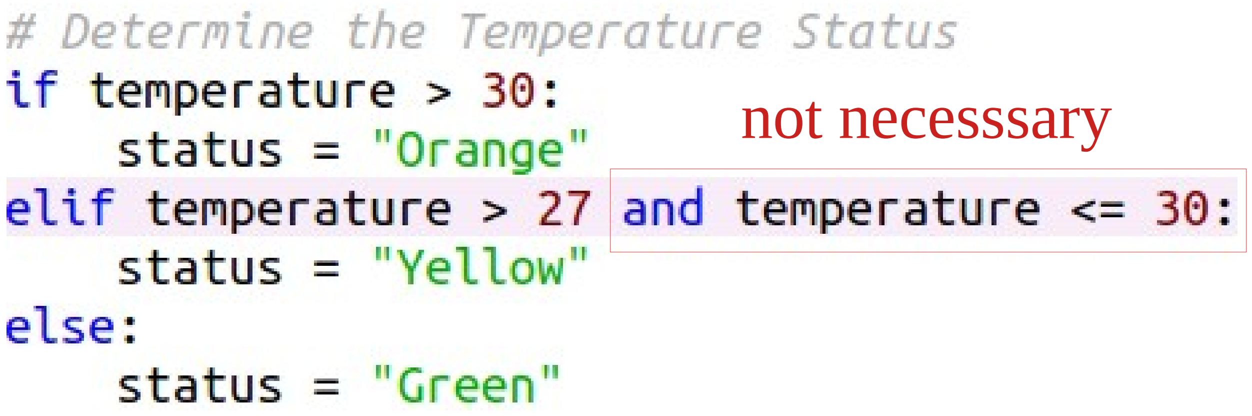
```
Enter Celsius Temperature: 18
Temperature Status: Green
```

Notice how, by taking the values in order – in this case, highest to lowest:



```
if temperature > 30:
    status = "Orange"
elif temperature > 27:
    status = "Yellow"
else:
    status = "Green"
```

there is no need to implement a combined condition using `and`:



```
# Determine the Temperature Status
if temperature > 30:
    status = "Orange"
elif temperature > 27 and temperature <= 30:
    status = "Yellow"
else:
    status = "Green"
```

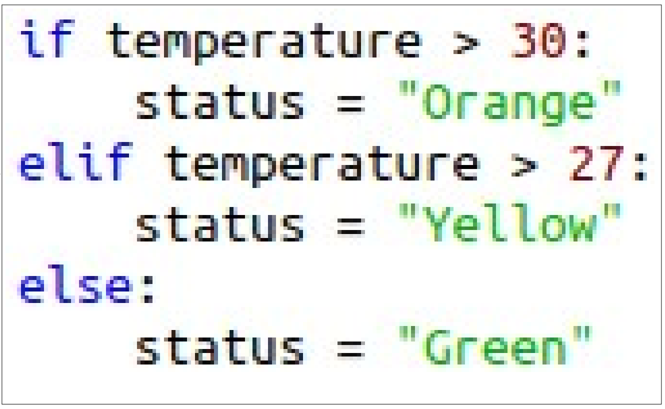
not necesssary

In the original version, if the temperature is not greater than 30, then the statement

```
    elif temperature > 27
```

is processed.

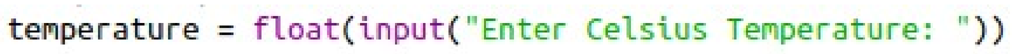
This `elif` can only be reached if the temperature is less than or equal to 30, and therefore there is no need to explicitly check for it:



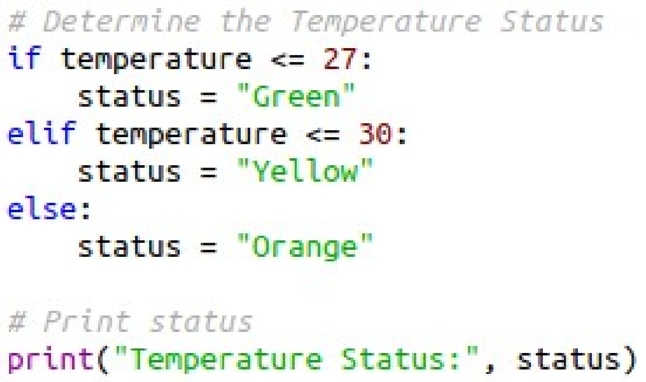
```
if temperature > 30:
    status = "Orange"
elif temperature > 27:
    status = "Yellow"
else:
    status = "Green"
```

Alternative Version

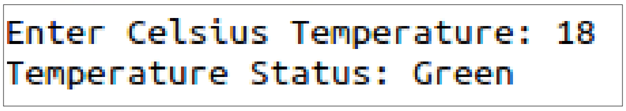
You can work from the smallest value to the largest:



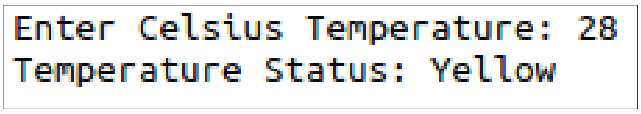
```
temperature = float(input("Enter Celsius Temperature: "))
```



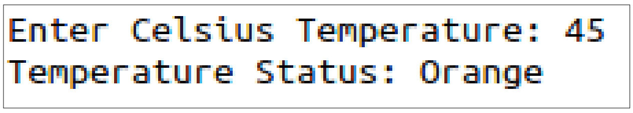
```
# Determine the Temperature Status
if temperature <= 27:
    status = "Green"
elif temperature <= 30:
    status = "Yellow"
else:
    status = "Orange"

# Print status
print("Temperature Status:", status)
```



```
Enter Celsius Temperature: 18
Temperature Status: Green
```



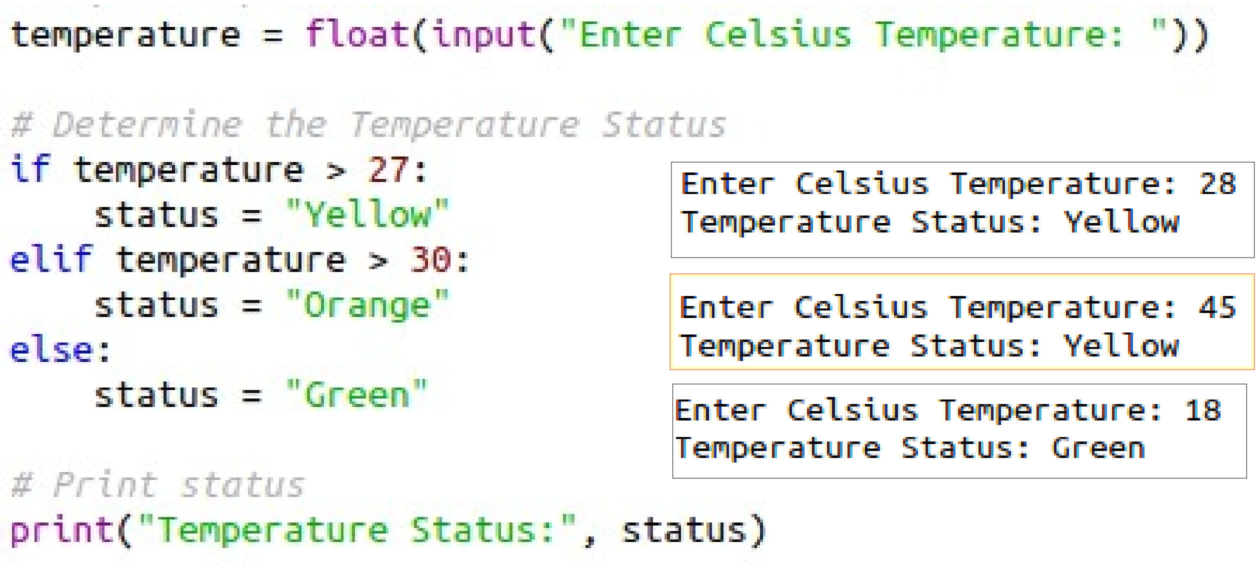
```
Enter Celsius Temperature: 28
Temperature Status: Yellow
```



```
Enter Celsius Temperature: 45
Temperature Status: Orange
```

Incorrect Version!

However, you must process the range of values in order:



```
temperature = float(input("Enter Celsius Temperature: "))

# Determine the Temperature Status
if temperature > 27:
    status = "Yellow"
elif temperature > 30:
    status = "Orange"
else:
    status = "Green"

# Print status
print("Temperature Status:", status)
```

```
Enter Celsius Temperature: 28
Temperature Status: Yellow
```

```
Enter Celsius Temperature: 45
Temperature Status: Yellow
```

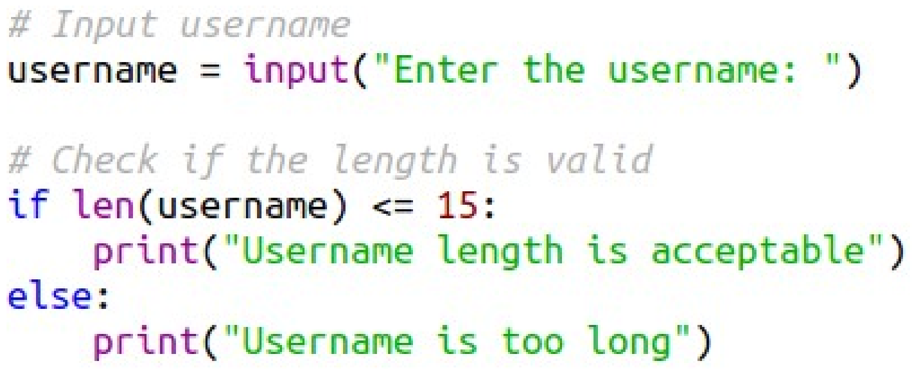
```
Enter Celsius Temperature: 18
Temperature Status: Green
```

In this example, an incorrect Yellow temperature status is reported for a temperature greater than 45, because the condition `temperature > 27` evaluates as `True` before program can check for a temperature greater than 30.

Decision Making with Strings

Decisions can also be implemented with string data.

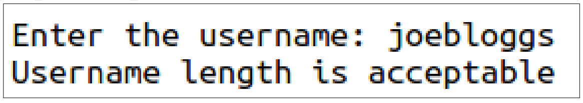
For example, the username for a specific Social Media account is required to have no more than 15 characters. The following program inputs a username and then displays a message indicating whether or not the length of the username is suitable.



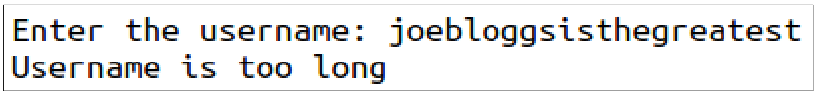
```
# Input username
username = input("Enter the username: ")

# Check if the length is valid
if len(username) <= 15:
    print("Username length is acceptable")
else:
    print("Username is too long")
```

The `len()` function checks the number of characters in the username string.

Sample Output

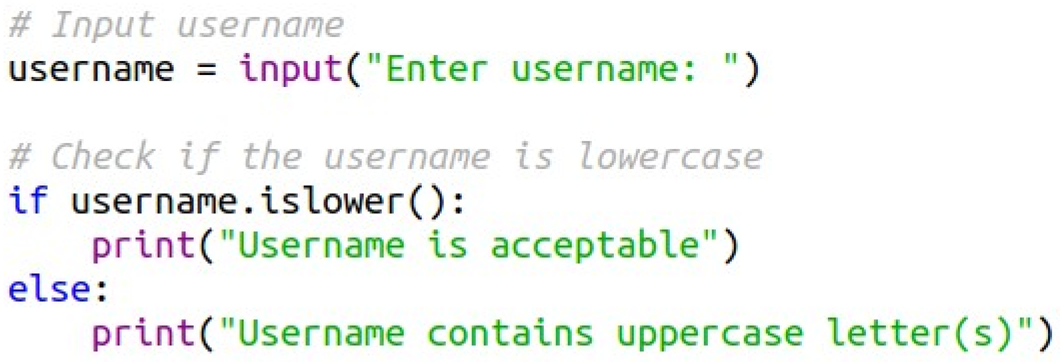


```
Enter the username: joebloggs
Username length is acceptable
```



```
Enter the username: joebloggsisthegreatest
Username is too long
```

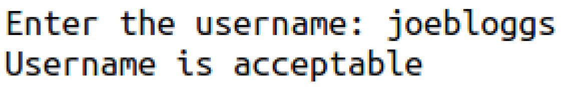
Another possible restriction on a username is that the characters must not contain any uppercase letters. This can be checked for using the string method `islower()`, which returns `True` if all letters in the string are lowercase, `False` otherwise (i.e. if there are any uppercase characters).



```
# Input username
username = input("Enter username: ")

# Check if the username is lowercase
if username.islower():
    print("Username is acceptable")
else:
    print("Username contains uppercase letter(s)")
```

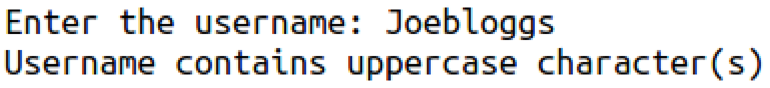
Sample Output

The program correctly identifies a valid username:



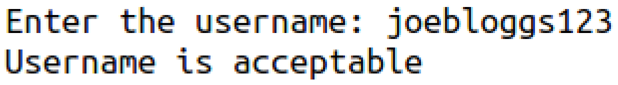
```
Enter the username: joebloggs
Username is acceptable
```

and it correctly identifies an invalid username:



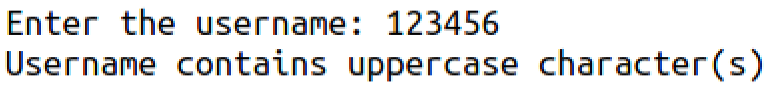
```
Enter the username: Joebloggs
Username contains uppercase character(s)
```

It will also permit non-letter characters:



```
Enter the username: joebloggs123
Username is acceptable
```

but will display an incorrect message if there are no lowercase characters in the string



```
Enter the username: 123456
Username contains uppercase character(s)
```

This is because the `islower()` method returns `False` if there are no lowercase characters in the string.

A correct implementation of username validation will be presented in Section 2(b) “Python's for loop”.

Nested Ifs

A block of code corresponding to an `if`, `elif` or `else` can contain any valid statements. This means you can include a `if-elif-else` structure within an `if`, `elif` or `else`. This is called a *Nested If*.

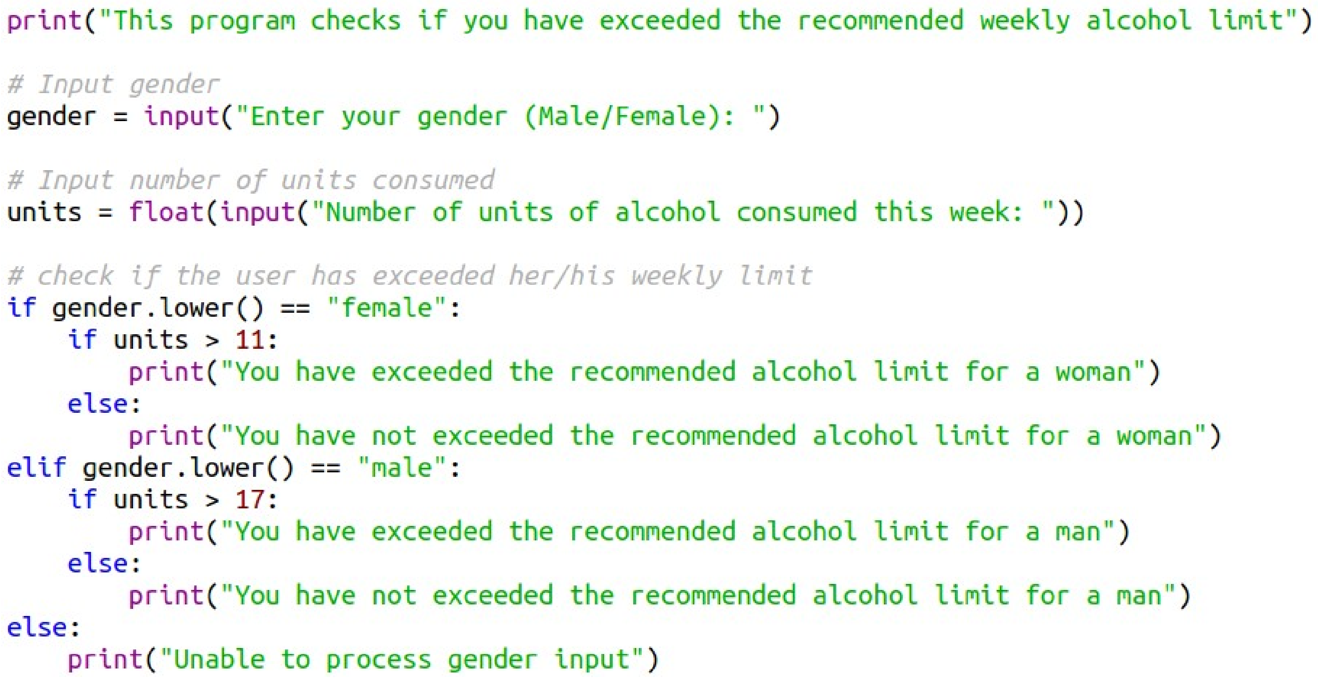
Example

For example, the HSE specifies low-risk drinking guidelines for women and men, as follows:

Gender	Limit
Female	Up to 11 standard drinks in a week
Male	Up to 17 standard drinks in a week

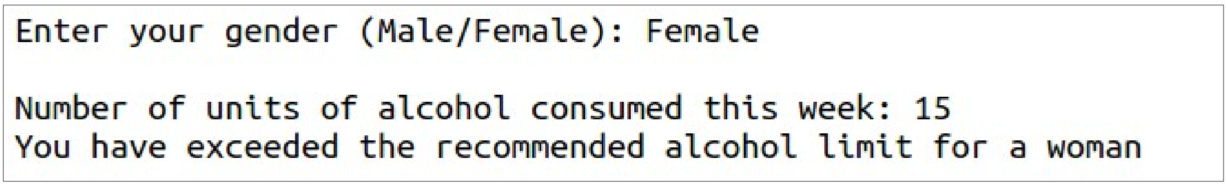
The following program uses nested ifs to check if the number of units consumed exceeds the recommended limit.

Python Program

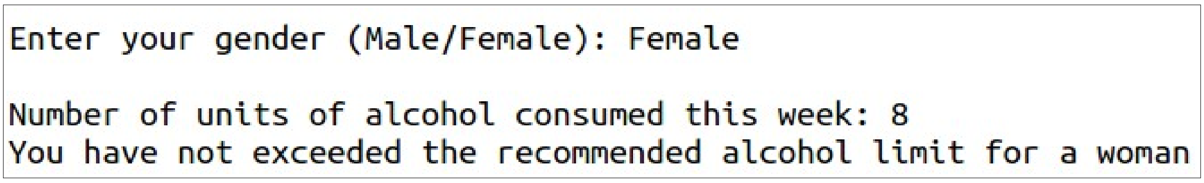


```python
print("This program checks if you have exceeded the recommended weekly alcohol limit")

# Input gender
gender = input("Enter your gender (Male/Female): ")

# Input number of units consumed
units = float(input("Number of units of alcohol consumed this week: "))

# check if the user has exceeded her/his weekly limit
if gender.lower() == "female":
    if units > 11:
        print("You have exceeded the recommended alcohol limit for a woman")
    else:
        print("You have not exceeded the recommended alcohol limit for a woman")
elif gender.lower() == "male":
    if units > 17:
        print("You have exceeded the recommended alcohol limit for a man")
    else:
        print("You have not exceeded the recommended alcohol limit for a man")
else:
    print("Unable to process gender input")
```

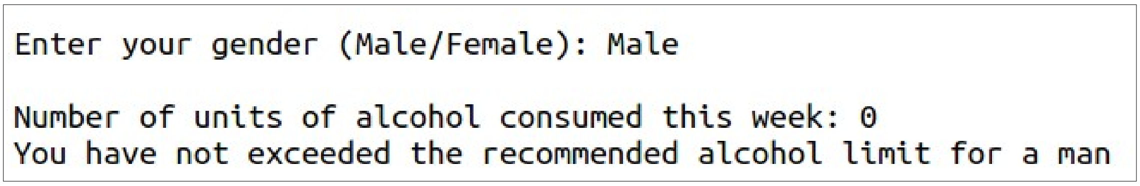
Sample Output (Program Executed 4 Times)



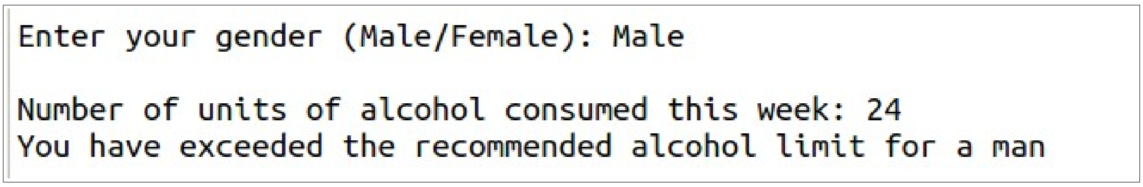
```
Enter your gender (Male/Female): Female

Number of units of alcohol consumed this week: 15
You have exceeded the recommended alcohol limit for a woman
```



```
Enter your gender (Male/Female): Female

Number of units of alcohol consumed this week: 8
You have not exceeded the recommended alcohol limit for a woman
```

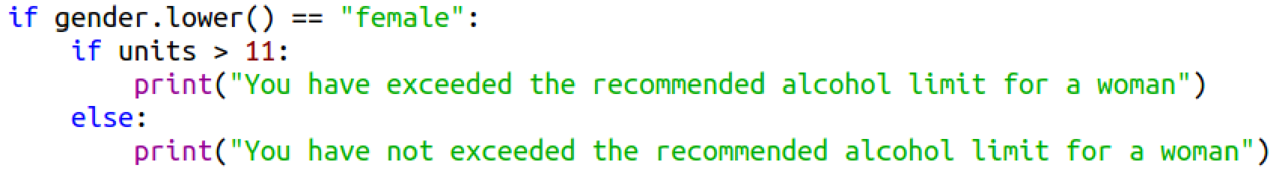


```
Enter your gender (Male/Female): Male

Number of units of alcohol consumed this week: 0
You have not exceeded the recommended alcohol limit for a man
```



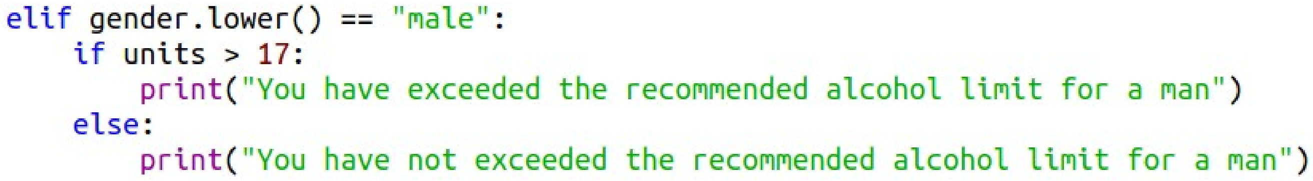
```
Enter your gender (Male/Female): Male

Number of units of alcohol consumed this week: 24
You have exceeded the recommended alcohol limit for a man
```

The program demonstrates an `if-else` structure within an `if`:



```
if gender.lower() == "female":
    if units > 11:
        print("You have exceeded the recommended alcohol limit for a woman")
    else:
        print("You have not exceeded the recommended alcohol limit for a woman")
```

as well as an `if-else` structure within an `else`:



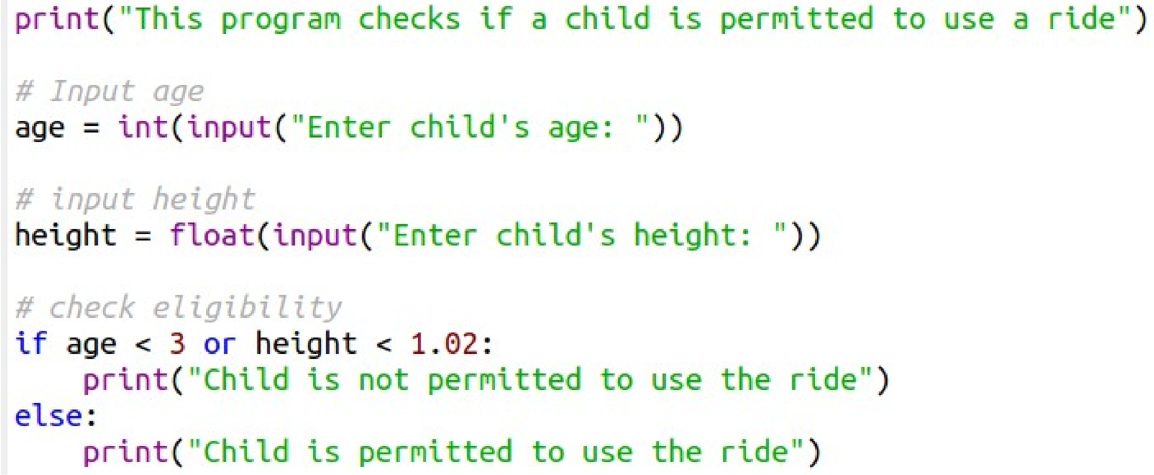
```
elif gender.lower() == "male":
    if units > 17:
        print("You have exceeded the recommended alcohol limit for a man")
    else:
        print("You have not exceeded the recommended alcohol limit for a man")
```

Decision Making with `and or not`

You can implement more complex decision using the logical operators `and or not` to combine and/or negate conditions.

For example, some amusement parks have age and height restrictions for their rides: children younger than 3 years old, or less than 1.02 metres in height, may not use the ride.

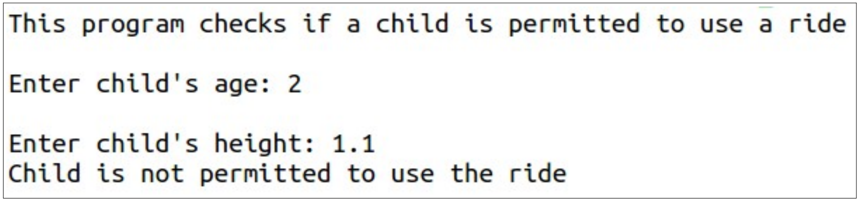
Python Program



```
print("This program checks if a child is permitted to use a ride")

# Input age
age = int(input("Enter child's age: "))

# input height
height = float(input("Enter child's height: "))

# check eligibility
if age < 3 or height < 1.02:
    print("Child is not permitted to use the ride")
else:
    print("Child is permitted to use the ride")
```

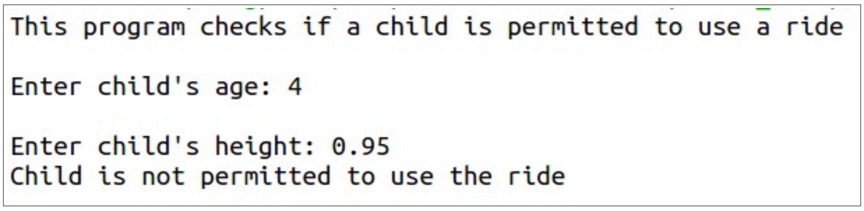
Sample Output

Too young:



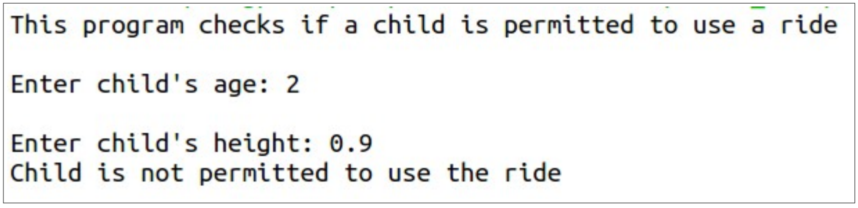
```
This program checks if a child is permitted to use a ride

Enter child's age: 2

Enter child's height: 1.1
Child is not permitted to use the ride
```

Too small:



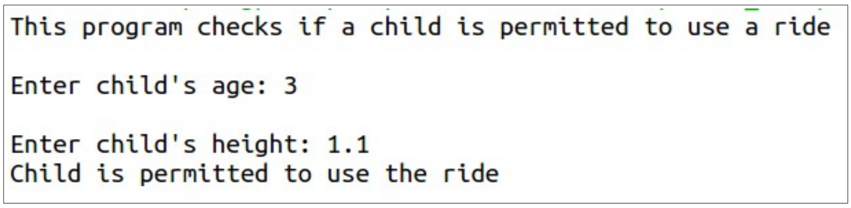
```
This program checks if a child is permitted to use a ride

Enter child's age: 4

Enter child's height: 0.95
Child is not permitted to use the ride
```

Too young and too small:



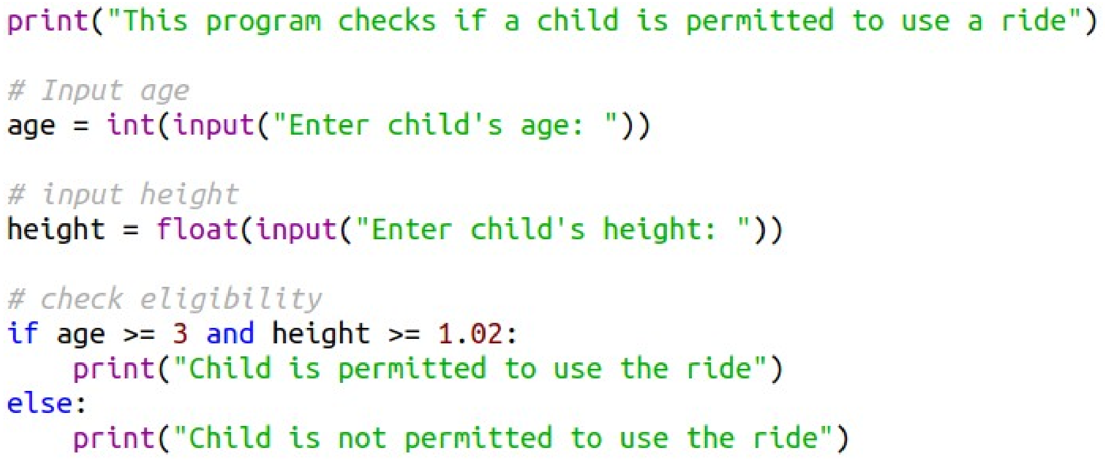
```
This program checks if a child is permitted to use a ride

Enter child's age: 2

Enter child's height: 0.9
Child is not permitted to use the ride
```

Permitted:



```
This program checks if a child is permitted to use a ride

Enter child's age: 3

Enter child's height: 1.1
Child is permitted to use the ride
```

Another way of implementing this program is using `and`



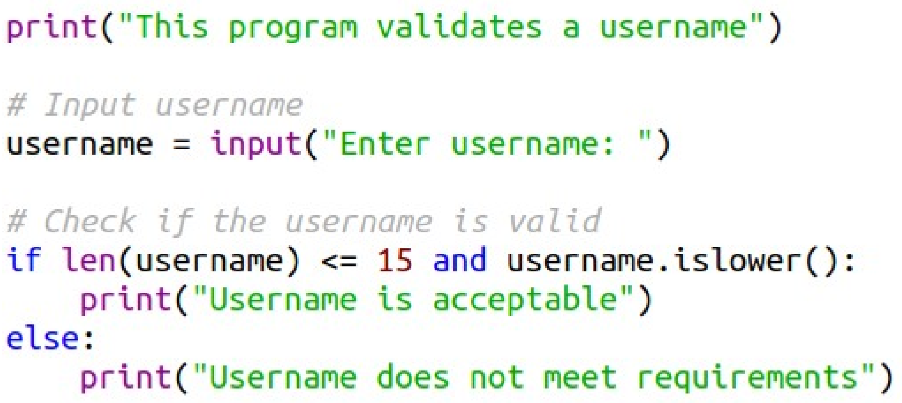
```
print("This program checks if a child is permitted to use a ride")

# Input age
age = int(input("Enter child's age: "))

# input height
height = float(input("Enter child's height: "))

# check eligibility
if age >= 3 and height >= 1.02:
    print("Child is permitted to use the ride")
else:
    print("Child is not permitted to use the ride")
```

In this case, a child is permitted to use the ride if s/he is at least 3 years old and at least 1.02m in height.

Example: Username Validation

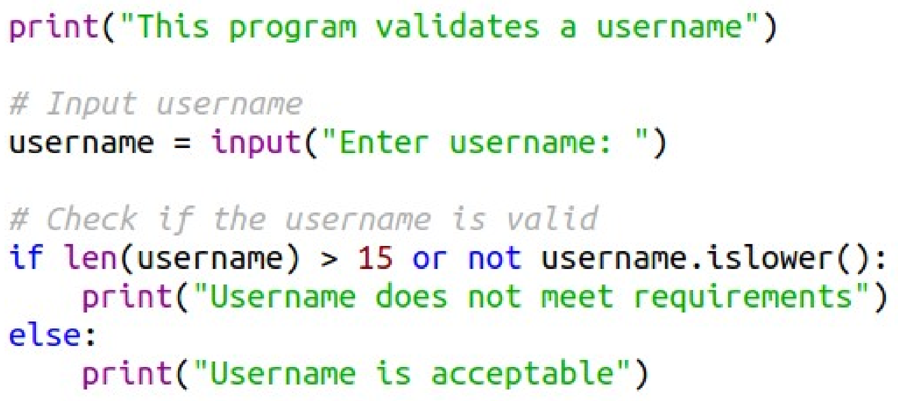
Two earlier examples separately demonstrated decision making using strings: the length of a username needed to be at most 15 characters, and no uppercase character were permitted. The two restrictions can be combined using `and` or `or`:

Version 1: Using `and`



```
print("This program validates a username")

# Input username
username = input("Enter username: ")

# Check if the username is valid
if len(username) <= 15 and username.islower():
    print("Username is acceptable")
else:
    print("Username does not meet requirements")
```

Version 2: Using `or`



```
print("This program validates a username")

# Input username
username = input("Enter username: ")

# Check if the username is valid
if len(username) > 15 or not username.islower():
    print("Username does not meet requirements")
else:
    print("Username is acceptable")
```

Notice how, when the `and` switches to `or`, and vice versa, the individual conditions are reversed:

Example	and		or	
Ride Restriction	age >= 3	height >= 1.02	age < 3	Height < 1.02
Username	len(username) <= 15	username.islower()	len(username) > 15	not username.islower()