**Python Conditional Statements.**

**Overview**

|  |
| --- |
| Decision making is required when we want to execute a code only if a certain condition is satisfied. The if ...elif ..else statement is used in Python for decision making. |

In Python you can define a series of conditionals using **if** for the first one, **elif** for the rest, up until the final (optional) **else** for anything not caught by other conditionals.

**The if statement**

We’ll start looking at the most basic type of an **if** statement. In its simplest form, it looks like this.



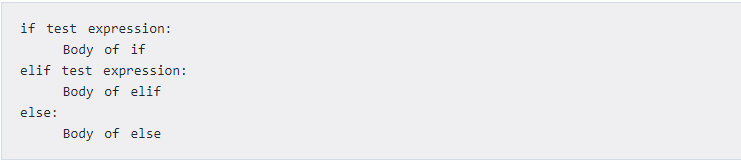
In the diagram shown above:

* **<expr>** is an expression evaluated in a Boolean context.
* **<statement>** is a valid Python statement, which must be indented. (You will see why very soon).

If **<expr>** is **true** (evaluates to a value that is “truthy”), then **<statement>** is executed. If **<expr>** is **false**, **<statement>** is skipped over and not executed.

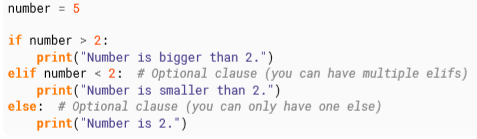
Note that the colon (:) following **<expr>** is required. Some programming languages required <expr> to be enclosed in parentheses, but Python does not.

The full syntax of **if...elif ...else** looks like the diagram below:



The **elif** is short for ***else if***. It allows us to check for multiple expressions. If the condition for **if** is False, it checks the condition of the next **elif** block so on. If all the conditions are ***False***, body of ***else*** is executed. The ***if*** block can have only one **else** block. But it can have multiple **elif** blocks.

Here are several examples of this type of if statement:



In the program above, we initialize a number to be equal to 5. Then we check if the number is greater than 2. If it is we print “Number is bigger than 2.”. We go again and check if the number is less than 2, we print “Number is smaller than 2.” If the number is neither greater than 2 or less than 2, then we print “Number is 2.”. Keep changing the value of the variable (number) and run the program to see different results.

One thing to note is that the ***elif***and ***else***statements must also have a colon at the end of the logical line followed by their corresponding block of statements (with proper indentation, of course). Remember that the ***elif*** and ***else*** parts are optional.

**Truth Values**

The following values are considered falsey, in that they evaluate to False when applied to a boolean operator:

* None
* False
* 0, or any numerical value equivalent to zero, for example 0, 0.0, 0j
* Empty sequences: ‘ ’, “ ”, ( ), [ ]
* Empty mappings: { }
* User defined types where the \_\_bool\_\_ or \_\_len\_\_ methods return 0 or false.

All other values in python evaluate to True.

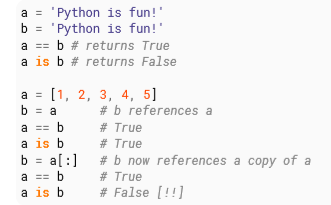
**Comparison by “is” vs “==”**

A common pitfall in Python is confusing the equality comparison operators **is** and **==**.

***a* == *b*** compares the value of *a* and *b*.

***a* is *b*** will compare the identities of *a* and *b.*

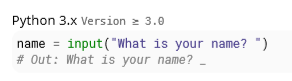
Let's illustrate with some code statements below:

****

Basically, *i****s*** can be thought of as shorthand for id(**a**) == id(**b**). Beyond this, there are quirks of the run-time environment that further complicated things. See page 84 of the **Handbook**

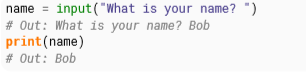
**User Input**

To get input from the user, use the **input** function(note: in python2.x the function is called **raw\_input** instead).

****

The function (**input**) takes a string argument, which displays it as a prompt and returns a string. The above code provides a prompt, waiting for the user to input.

If the user types “Bob” and hits enter, the variable name will be assigned to the string “Bob”:



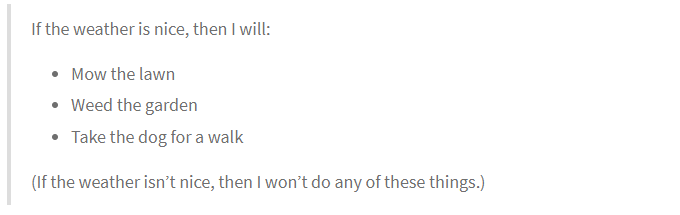
Note that the input is always of type **str**, which is important if you want the user to enter strings. To convert the input into a different type, you will have to perform type conversion as shown below.



It’s recommended to use ***try/except*** (we will cover this later on) blocks to catch exceptions (occurs during the execution of a program that disrupts the normal flow of the program instructions) when dealing with user inputs.

**Grouping Statements: indentation and Blocks**

Let's say you want to evaluate a condition and then do more than one thing if it is true. Have a look at the diagram below:



In all the examples shown above, each if <expr>: has been followed by only a single <statement>.. There needs to be some way to say “if <expr> is true, do all of the following things.”

**Does indenting matter in Python.**

In most other programming languages, indentation is used to make a code look pretty and readable. But in python, it is required for indicating what block of code a statement belongs to. Consider the following code below:



The lines print('Logging on ...') and print('Incorrect password.') are two separate code blocks. These ones happen to be only a single line long, but Python lets you write code blocks consisting of any number of statements.

**Compulsory Task : Midrand Speedster.**

****

You are tasked by JHB Metro Police to write a program for their new Camera Driver Demerit System.

The program should ask the driver to give you their current speed in km/h and the average allowed speed of the road. The program should conform to the following requirements:

1. If the speed is less than 60, it should print “**OK**”.
2. Otherwise, for every 5km above the speed limit ( e.g 60), it should give the driver one demerit point and print the total number of demerit points. For example, if the speed is 70km/h and the location’s average is 60km/h, it should print: “**Points: 2**”.
3. If the driver gets more than 12 demerit points, the function should print “**Time to go to jail!**”

**Sample input:**



**Sample output:**

