# If Statements

## Simple Conditions

The statements introduced in this chapter will involve tests or conditions. More syntax for conditions will be introduced later, but for now consider simple arithmetic comparisons that directly translate from math into Python. Try each line separately in the Shell

2 < 5

3 > 7

x = 11

x > 10

2 \* x < x

type(**True**)

You see that conditions are either True or False (with no quotes!). These are the only possible Boolean values (named after 19th century mathematician George Boole). In Python the name Boolean is shortened to the type bool. It is the type of the results of true-false conditions or tests.

## Simple if Statements

Run this example program, suitcase.py. Try it at least twice, with inputs: 30 and then 55. As you an see, you get an extra result, depending on the input. The main code is:

weight = float(input("How many pounds does your suitcase weigh? "))

**if** weight > 50:

print("There is a $25 charge for luggage that heavy.")

print("Thank you for your business.")

The middle two line are an if statement. It reads pretty much like English. If it is true that the weight is greater than 50, then print the statement about an extra charge. If it is not true that the weight is greater than 50, then don’t do the indented part: skip printing the extra luggage charge. In any event, when you have finished with the if statement (whether it actually does anything or not), go on to the next statement that is not indented under the if. In this case that is the statement printing “Thank you”.

The general Python syntax for a simple if statement is

if condition :

indentedStatementBlock

If the condition is true, then do the indented statements. If the condition is not true, then skip the indented statements.

Another fragment as an example:

**if** balance < 0:

transfer = -balance

*# transfer enough from the backup account:*

backupAccount = backupAccount - transfer

balance = balance + transfer

As with other kinds of statements with a heading and an indented block, the block can have more than one statement. The assumption in the example above is that if an account goes negative, it is brought back to 0 by transferring money from a backup account in several steps.

In the examples above the choice is between doing something (if the condition is True) or nothing (if the condition is False). Often there is a choice of two possibilities, only one of which will be done, depending on the truth of a condition.

## if-else Statements

Run the example program, clothes.py. Try it at least twice, with inputs 50 and then 80. As you can see, you get different results, depending on the input. The main code of clothes.py is:

temperature = float(input('What is the temperature? '))

**if** temperature > 70:

print('Wear shorts.')

**else**:

print('Wear long pants.')

print('Get some exercise outside.')

The middle four lines are an if-else statement. Again it is close to English, though you might say “otherwise” instead of “else” (but else is shorter!). There are two indented blocks: One, like in the simple if statement, comes right after the if heading and is executed when the condition in the if heading is true. In the if-else form this is followed by an else: line, followed by another indented block that is only executed when the original condition is false. In an if-else statement exactly one of two possible indented blocks is executed.

A line is also shown **out**dented next, about getting exercise. Since it is outdented, it is not a part of the if-else statement: It is always executed in the normal forward flow of statements, after the if-else statement (whichever block is selected).

The general Python if-else syntax is

if condition :

indentedStatementBlockForTrueCondition

else:

indentedStatementBlockForFalseCondition

These statement blocks can have any number of statements, and can include about any kind of statement.

See [Graduate Exercise](http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html#graduateex)

## More Conditional Expressions

All the usual arithmetic comparisons may be made, but many do not use standard mathematical symbolism, mostly for lack of proper keys on a standard keyboard.

| **Meaning** | **Math Symbol** | **Python Symbols** |
| --- | --- | --- |
| Less than | < | < |
| Greater than | > | > |
| Less than or equal | ≤ | <= |
| Greater than or equal | ≥ | >= |
| Equals | = | == |
| Not equal | ≠ | != |

There should not be space between the two-symbol Python substitutes.

Notice that the obvious choice for equals, a single equal sign, is not used to check for equality. An annoying second equal sign is required. This is because the single equal sign is already used for assignment in Python, so it is not available for tests.

**Warning**

It is a common error to use only one equal sign when you mean to test for equality, and not make an assignment!

Tests for equality do not make an assignment, and they do not require a variable on the left. Any expressions can be tested for equality or inequality (!=). They do not need to be numbers! Predict the results and try each line in the Shell:

x = 5

x

x == 5

x == 6

x

x != 6

x = 6

6 == x

6 != x

'hi' == 'h' + 'i'

'HI' != 'hi'

[1, 2] != [2, 1]

An equality check does not make an assignment. Strings are case sensitive. Order matters in a list.

Try in the Shell:

'a' > 5

When the comparison does not make sense, an Exception is caused. [[1]](http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html#id13)

Following up on the discussion of the inexactness of float arithmetic in [String Formats for Float Precision](http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/float.html#precision-formats), confirm that Python does not consider .1 + .2 to be equal to .3: Write a simple condition into the Shell to test.

Here is another example: Pay with Overtime. Given a person’s work hours for the week and regular hourly wage, calculate the total pay for the week, taking into account overtime. Hours worked over 40 are overtime, paid at 1.5 times the normal rate. This is a natural place for a function enclosing the calculation.

Read the setup for the function:

**def** calcWeeklyWages(totalHours, hourlyWage):

*'''Return the total weekly wages for a worker working totalHours,*

*with a given regular hourlyWage. Include overtime for hours over 40.*

*'''*

The problem clearly indicates two cases: when no more than 40 hours are worked or when more than 40 hours are worked. In case more than 40 hours are worked, it is convenient to introduce a variable overtimeHours. You are encouraged to think about a solution before going on and examining mine.

You can try running my complete example program, wages.py, also shown below. The format operation at the end of the main function uses the floating point format ([String Formats for Float Precision](http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/float.html#precision-formats)) to show two decimal places for the cents in the answer:

**def** calcWeeklyWages(totalHours, hourlyWage):

*'''Return the total weekly wages for a worker working totalHours,*

*with a given regular hourlyWage. Include overtime for hours over 40.*

*'''*

**if** totalHours <= 40:

totalWages = hourlyWage\*totalHours

**else**:

overtime = totalHours - 40

totalWages = hourlyWage\*40 + (1.5\*hourlyWage)\*overtime

**return** totalWages

**def** main():

hours = float(input('Enter hours worked: '))

wage = float(input('Enter dollars paid per hour: '))

total = calcWeeklyWages(hours, wage)

print('Wages for {hours} hours at ${wage:.2f} per hour are ${total:.2f}.'

.format(\*\*locals()))

main()

Here the input was intended to be numeric, but it could be decimal so the conversion from string was via float, not int.

Below is an equivalent alternative version of the body of calcWeeklyWages, used in wages1.py. It uses just one general calculation formula and sets the parameters for the formula in the if statement. There are generally a number of ways you might solve the same problem!

**if** totalHours <= 40:

regularHours = totalHours

overtime = 0

**else**:

overtime = totalHours - 40

regularHours = 40

**return** hourlyWage\*regularHours + (1.5\*hourlyWage)\*overtime

## Multiple Tests and if-elif Statements

Often you want to distinguish between more than two distinct cases, but conditions only have two possible results, True or False, so the only direct choice is between two options. As anyone who has played “20 Questions” knows, you can distinguish more cases by further questions. If there are more than two choices, a single test may only reduce the possibilities, but further tests can reduce the possibilities further and further. Since most any kind of statement can be placed in an indented statement block, one choice is a further if statement. For instance consider a function to convert a numerical grade to a letter grade, ‘A’, ‘B’, ‘C’, ‘D’ or ‘F’, where the cutoffs for ‘A’, ‘B’, ‘C’, and ‘D’ are 90, 80, 70, and 60 respectively. One way to write the function would be test for one grade at a time, and resolve all the remaining possibilities inside the next else clause:

**def** letterGrade(score):

**if** score >= 90:

letter = 'A'

**else**: *# grade must be B, C, D or F*

**if** score >= 80:

letter = 'B'

**else**: *# grade must be C, D or F*

**if** score >= 70:

letter = 'C'

**else**: *# grade must D or F*

**if** score >= 60:

letter = 'D'

**else**:

letter = 'F'

**return** letter

This repeatedly increasing indentation with an if statement as the else block can be annoying and distracting. A preferred alternative in this situation, that avoids all this indentation, is to combine each else and if block into an elif block:

**def** letterGrade(score):

**if** score >= 90:

letter = 'A'

**elif** score >= 80:

letter = 'B'

**elif** score >= 70:

letter = 'C'

**elif** score >= 60:

letter = 'D'

**else**:

letter = 'F'

**return** letter

The most elaborate syntax for an if-elif-else statement is indicated in general below:

if condition1 :

indentedStatementBlockForTrueCondition1

elif condition2 :

indentedStatementBlockForFirstTrueCondition2

elif condition3 :

indentedStatementBlockForFirstTrueCondition3

elif condition4 :

indentedStatementBlockForFirstTrueCondition4

else:

indentedStatementBlockForEachConditionFalse

The if, each elif, and the final else line are all aligned. There can be any number of elif lines, each followed by an indented block. (Three happen to be illustrated above.) With this construction exactly one of the indented blocks is executed. It is the one corresponding to the first True condition, or, if all conditions are False, it is the block after the final else line.

Be careful of the strange Python contraction. It is elif, not elseif. A program testing the letterGrade function is in example program grade1.py.

See [Grade Exercise](http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/ifstatements.html#gradeex).

A final alternative for if statements: if-elif-.... with no else. This would mean changing the syntax for if-elif-else above so the final else: and the block after it would be omitted. It is similar to the basic if statement without an else, in that it is possible for no indented block to be executed. This happens ifnone of the conditions in the tests are true.

With an else included, exactly one of the indented blocks is executed. Without an else, at most one of the indented blocks is executed.

**if** weight > 120:

print('Sorry, we can not take a suitcase that heavy.')

**elif** weight > 50:

print('There is a $25 charge for luggage that heavy.')

This if-elif statement only prints a line if there is a problem with the weight of the suitcase.

### Sign Exercise

Write a program sign.py to ask the user for a number. Print out which category the number is in: 'positive', 'negative', or 'zero'.

### Even Print Exercise

Write a program even1.py with a function printEven with heading:

**def** printEven(nums):

*'''Given a list of integers nums,*

*print the even ones.*

*>>> printEven([4, 1, 3, 2, 7])*

*4*

*2*

*'''*

In your main program, test the function, calling it several times with different lists of integers. Hint: A number is even if its remainder, when dividing by 2, is 0.

### Even List Exercise

Write a program even2.py with a function chooseEven with heading:

**def** chooseEven(nums):

*'''Given a list of integers, nums,*

*return a list containing only the even ones.*

*>>> chooseEven([4, 1, 3, 2, 7])*

*[4, 2]*

*'''*

In your main program, test the function, calling it several times with different lists of integers and printing the results. Hint: Create a new list, and append the appropriate numbers to it.