Wiki - Level 3 - Logic and conditionals

Example: “True_False”

The programs that we are capable of constructing in our subset of Python are still fairly uninteresting. In particular, the sequence of statements executed by Python does not depend on the actual values used in the computation. In language level three, we will extend the types of data that we consider to include a new binary data type called Booleans. (The corresponding type in Python is bool.) The two Boolean values are True and False. We begin by noting that Python programs can print and assign Boolean values in the same manner as numbers and strings. For example,

```
print True, False
```

prints out the Boolean values True and False as expected.

Examples: “boolean_expressions”, “is_positive”

Boolean values can be combined using the three Boolean operators and, or and not. The example “boolean_expressions” contains several examples of Boolean expressions. Boolean values can also be created by comparing other types of data. In particular, we will often compare numbers using relational comparisons such as ==, !=, < and >. These operators take two numbers and return a Boolean value. (Relation comparisons also work on strings.) The example “is_positive” defines a simple function that returns True if an input number is positive and False otherwise.

Example: “absolute_value”

The main use of Boolean values is to control the sequence in which Python executes statements. For now, we will focus on the simplest type of conditional statement: the if statement. An if statement consists of an if clause followed by zero or more elif clauses followed by an optional else clause. In Python, if clauses have the form

```
if condition:
   body
```

Note that body of the if clause corresponds to a sequence of indented Python statements (similar to the body of a function definition). You may use the left and right arrow keys to explore the structure of the if clause. To execute an if statement, Python first evaluates the Boolean expression that forms the condition of the ifclause. If the condition evaluates to True, Python executes the body of the if clause. If the condition evaluates to False, the body is ignored.

An else clause has the form:

```
else:
   body
```

else clauses (when used) always follow either an if clause or an elif clause. The body of the else clause also consists of a sequence of indented Python statements. The body of the else clause is executed its preceding if clause's condition evaluates to False. The example “absolute_value” illustrates the use of an if and an else clause. Note that *Pystep* first evaluates the condition for the if clause and then replaces the if and else clauses by the appropriate body.

Examples: “sign”, “military_time”

In many situations, we wish to select among more than two choices. For this situation, we suggest using one or more elif clauses in conjunction with if and else clauses. An elif clause has the form:

```
elif condition:
    body
```

elif clauses always follow either an if clause or another elif clause. If the conditions for the preceding sequence of if and elif clauses all evaluate to False, the condition for the elif is evaluated. If the condition evaluates to True, the body of the clause is executed. Otherwise, any following elif and else clauses are then executed. The example “sign” uses an if-elif-else sequence to return -1 when an input number is negative, 0 when the number is zero and 1 when the number is positive. The example “military_time” illustrates the use of a longer chain of conditionals to convert military time to twelve-hour time.

Example: “factorial”

The final “factorial” example demonstrates that the combination of functions and conditionals is surprisingly powerful. The factorial function in this example calls itself conditionally based on the value of its input. This idea of a function calling itself is known as recursion. While we won't leverage the power of recursion in this course, recursion is a critical tool in many more advanced computer applications.