

Python Reference

General Program Structure

```
def identifier ( ):
    statement(s)
identifier ( )
```

Note that the identifier in the header must be the same as the one in the call (the last line).

Documentation

The pound sign (#) is used to include comments to document code. The python interpreter ignores everything from the pound sign to the end of the line.

Storing a value in memory (Assignment)

```
identifier = expression
```

identifier is the name of a variable that represents a memory address; *expression* is a value or a set of values and operators that evaluate to a value. Examples:

```
y = 7
x = 12 + y
a = "Hello " + "World"
```

Naming Variables

The rules for making identifiers are:

- 1) Use characters “a” to “z,” “A” to “Z,” “0” to “9” and “_” to make the names. No other characters, including spaces may be used in an identifier.
- 2) The first symbol may not come from “0” to “9.”
- 3) The identifier may not be the same as a python reserved word. Note that the identifier *sprint* is fine even though it contains *print*.

Identifiers should be descriptive; the name of a variable normally will be a noun or object-noun combination such as *total* or *overtimePay*.

By convention, programmers usually begin the names of variables with a lowercase letter.

Arithmetic Expressions

An arithmetic expression is a series of operands separated by operators. The operands may be literals (numbers) or variables. Operands include +, -, *, /, and %.

Expressions are evaluated according to the standard order of operations. Enclosing an operation in parentheses increases its precedence. Spaces don't matter.

Two example expressions:

```
12 + 33 / 11 * 2 (evaluates to 18)    6 * ( y + z ) / w
```

Type Conversion

The result of any operation in which one of the operands is floating point is float. If all the operands are integers, the type is integer. Integer division is a common source of errors.

Converting integer to float: **float (value)**

Converting float to integer: **int (value)**

Converting either integer or float to string: **str (value)**

Input and Output

Python input statements work like an assignment. They include an identifier for a memory location, the assignment operator, and a call to the input function:

```
identifier = input ( prompt )  
identifier = raw_input ( prompt )
```

Here the *prompt* is an expression that evaluates to the string type. To obtain numeric input, use the plain input function; to get non-numeric input, use the raw_input function.

The print statement is used for output:

```
print                                # prints a blank line  
print expression  
print expression1, expression2, ...
```

Counter-controlled Iteration

The basic syntax for the header of a **for** block is:

```
for variable in list:
```

Variable is replaced by a variable that will successively take on values from the list with each iteration. *List* is either an explicit list of values enclosed in square brackets, or the list returned by a call to the range function. The number of values in the list is the number of times the block repeats.

Some examples:

```
for x in [ 1, 2, 3, 4, 5, 6, 7 ]:      # repeats seven times  
for day in [ "Mon", "Tues", "Wed", "Thurs", "Fri" ]:    # five times
```

The range function has three forms. **range (v)** returns a list with 0 as its first element and v-1 as its last element. **range (v1, v2)** returns a list with v1 as its first element and v2-1 as its last element. **range (v1, v2, step)** returns a list with v1 as its first element and subsequent elements determined by adding step to each previous element. The last element is the largest element generated that is less than v2.

The third form of the range function can be used to create lists in descending order.

Examples

```
range (5) returns [ 0, 1, 2, 3, 4 ]  
range ( 2, 8 ) returns [ 2, 3, 4, 5, 6, 7 ]  
range ( 2, 3, 9 ) returns [ 2, 5, 8 ]  
range ( 9, 2, -3 ) returns [ 9, 6, 3 ]
```

Sentinel (Condition) Controlled Iteration

If the programmer does not know how many times a block of statements must be repeated, the situation calls for condition-controlled iteration. Typical usages include repeating a task until there is no more data to process and repeating a task until some stopping condition (for instance, an investment doubling in value) is reached.

The syntax for this type of iteration is

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

Condition is an expression that evaluates to true or false, as in the case of the **if** structure. Typically the condition compares some variable to a value that changes when the iteration must stop. Examples include an input of nothing (or “quit”) for a name, or a read value of the end-of-file character from a file. The variable in this case is called a sentinel.

Note that one of the repeated statements must be able to change the value of the sentinel; otherwise, if control enters the loop, it will iterate forever.

The usual way to set up a condition-controlled iteration is as follows:

```
get initial value for sentinel variable  
while sentinel is not the halting value:  
    process information  
    get next value for sentinel
```

Formatting Output

Most programming languages include some method for forcing output into a specific format. Python provides *formatting strings* to allow programmers to specify how many characters to allot for some piece of information and also to specify the precision of floating-point information.

A formatting string appears inside quotes immediately before the variable it is applied to. The formatting string starts with a percent sign, and then has a number indicating how many characters to allow for the output, and then if the out value is floating point, a decimal followed by the number of places. The last character indicates the type of information (i, f or s). The variable follows the quotes, with another percent sign in front. Here are some examples:

```
“The average is %9.3f”%avg
```

This allows nine spaces, and shows three decimal places when showing the contents of the variable **avg**.

```
“Employee Name: %14s”%empName
```

This allocates 14 spaces for the variable **empName**, even if the name is less than 14 characters.

Note that the resulting data from using a formatting string is of the string type. Also be aware that if a value is wider than the specified width, it won’t be truncated. In the second instance above, if the name were 18 characters, they would all still be output.

Selection

Many problem solutions require the ability to perform some step or steps only under certain circumstances, or to choose between two or more alternate sets of steps. Selection structures provide the mechanism for encoding such choices in a program.

There are three general forms for building selection structures:

if condition :

statement(s) to perform

if condition :

statement(s) to perform if condition is true

else:

statement(s) to perform if condition is false

if condition :

statement(s) to perform if condition is true

elif second condition :

statement(s) to perform if second condition is true

etc.

else:

statement(s) to perform if none of the conditions is true

Conditions

Conditions are expressions that evaluate to either true or false. Generally conditions include comparison operators (> < >= <= == or !=).