Python Reference

Udacity cs101: Unit 1

Arithmetic Expressions

addition: *<Number>* **+** *<Number>*
outputs the sum of the two input numbers

multiplication: *<Number>* ***** *<Number>*
outputs the product of the two input numbers

subtraction: *<Number>* **-** *<Number>*
outputs the difference between the two input numbers

division: *<Number>* / *<Number>*
outputs the result of dividing the first number by the second
Note: if both numbers are whole numbers, the result is truncated to just the whole number part.

Variables and Assignment

Names

A *<Name>* in Python can be any sequence of letters, numbers, and underscores (_) that does not start with a number. We usually use all lowercase letters for variable names, but capitalization must match exactly. Here are some valid examples of names in Python (but most of these would not be good choices to actually use in your programs):

```
my_name
one2one
Dorina
this_is_a_very_long_variable_name
```

Assignment Statement

An assignment statement assigns a value to a variable:

<Name> **=** *<Expression>*

After the assignment statement, the variable *<Name>* refers to the value of the *<Expression>* on the right side of the assignment. An *<Expression>* is any Python construct that has a value.

Multiple Assignment

We can put more than one name on the left side of an assignment statement, and a corresponding number of expressions on the right side:

<*Name*$_1$>`,` <*Name*$_2$>`,` … `=` <*Expression*$_1$>`,` <*Expression*$_2$>`,` …

All of the expressions on the right side are evaluated first. Then, each name on the left side is assigned to reference the value of the corresponding expression on the right side. This is handy for swapping variable values. For example,

```
s, t = t, s
```

would swap the values of `s` and `t` so after the assignment statement `s` now refers to the previous value of `t`, and `t` refers to the previous value of `s`.

Strings

A string is sequence of characters surrounded by quotes. The quotes can be either single or double quotes, but the quotes at both ends of the string must be the same type. Here are some examples of strings in Python:

```
"silly"
'string'
"I'm a valid string, even with a single quote in the middle!"
```

String Concatenation

We can use the `+` operator on strings, but it has a different meaning than when it is used on numbers.

string concatenation: <*String*> `+` <*String*>
outputs the concatenation of the two input strings (pasting the string together with no space between them)

We can also use the multiplication operator on strings:

string multiplication: <*String*> `*` <*Number*>
outputs a string that is <*Number*> copies of the input <*String*> pasted together

Indexing Strings

The indexing operator provides a way to extract subsequences of characters from a string.

string indexing: <*String*>`[`<*Number*>`]`
outputs a single-character string containing the character at position <*Number*> of the input <*String*>. Positions in the string are counted starting from 0, so `s[1]` outputs the second character in `s`. If the <*Number*> is negative, positions are counted from the end of the string: `s[-1]` is the last character in `s`.

string extraction: *<String>*`[`*<Start Number>*`:`*<Stop Number>*`]`
outputs a string that is the subsequence of the input string starting from position *<Start Number>* and ending just before position *<Stop Number>*. If *<Start Number>* is missing, starts from the beginning of the input string; if *<Stop Number>* is missing, goes to the end of the input string.

Finding Subsequence in Strings

The `find` method provides a way to find sub-sequences of characters in strings.

find: *<search string>*`.find(`*<target string>*`)`
outputs a number giving the position in *<search string>* where *<target string>* first appears. If there is no occurrence of *<target string>* in *<search string>*, outputs `-1`.

To find later occurrences, we can also pass in a number to `find`:

find at or after: *<search string>*`.find(`*<target string>*`,` *<start number>*`)`
outputs a number giving the position in *<search string>* where *<target string>* first appears that is at or after the position give by *<start number>*. If there is no occurrence of *<target string>* in *<search string>* at or after *<start number>*, outputs `-1`.

Converting Numbers to Strings

(Introduced in the last homework question, not in the lecture.)

str: `str(`*<Number>*`)`
outputs a string that represents the input number. For example, `str(23)` outputs the string `'23'`.