Variables

Variables

Python receives information by means of variables. A variable is a dedicated piece of computer memory that holds some information. For example,

[2]:
```
a = 5
```

tells python to assign 5 (an integer number) to the variable with the variable name `a`. Note that the `=` here is used for variable assignment, it does not have the same meaning as the mathematical symbol ("assign-variable-to" rather than "is-equal-to").

If we wanted to access the value that our variable `a` holds, we can refer to it by it's name. For example, if we want to print the value to terminal:

[3]:
```
print(a)
```

```
5
```

Data Types

The information stored in memory needs to be interprated if it's to be of any use to us. To achieve this Python (and many other programming languages) uses variable types.

In the example above we used an integer or `int` type. In order to check what type a variable has, we can use the `type()` function:

[4]:
```
print(type(a))
```

```
<class 'int'>
```

The other basic variable types we will be working with are floating point numbers and strings.

Floating point numbers (or `float`) are numbers with decimal parts, for example:

[6]:
```
print(type(5.2))
```

```
<class 'float'>
```

Strings (or `str`), are a collection of unicode characters (letters, numbers, symbols, ect). Basically, the contents of any text file can be seen as a string. Strings are represented using parenthesis:

[7]:
```
print(type('This is a string.'))
```

```
<class 'str'>
```

You are not limited to single quotes. For single line strings you can use double quotes as well:

[8]:
```
print('String using single quotes, " does not break the string.')
print("String using double quotes, ' doesn't break the string.")
```

[8]:
```
"String using double qotes, ' doesn't break the string"
```

For strings containing line breaks, you can use `'''` or `"""`:

[2]:
```
print(
'''String with a
line break'''
)

print(
"""Another string with a
line break"""
)
```

```
String with a
line break
Another string with a
line break
```

You could also insert line breaks using a the special character `'\n`

Variable Names

So far we have been using single letters (`a`, `b`, `c`, `d`, …) as variable names, but this approach can be confusing for long segments of codes. Variable names should be as clear and descriptive as possible (describing what they are used for), while still being short enough to type out efficiently.

To this end we should delve into some of the restrictions on the character sequences that make up variable names: * The characters must all be letters, digits, or underscores (_) , and must start with a letter. In particular, punctuation and blanks are not allowed. * There are some words that are reserved for special use in Python. You may not use these words as your own identifiers. This is the full list:

False

class

finally

is

return

None

continue

for

lambda

try

True

def

from

nonlocal

while

and

del

global

not

with

as

if

elif

or

yield

assert

else

import

pass

break

except

in

raise

- Python is case sensitive: The variable names `last`, `LAST`, and `LaSt` are all different.

Now, you may want to use a variable that is more than one word long, for example `price at opening`, but blanks are illegal! One poor option is just leaving out the blanks, like `priceatopening`. Then it may be hard to figure out where words split. Two practical options are: * Underscore separated: putting underscores (which are legal) in place of the blanks, like `price_at_opening`. * Using camel-case: omitting spaces and using all lowercase, except capitalizing all words after the first, like `priceAtOpening`. * Using Pascal-case: similar to camel-case but capitalising the first word, `PriceAtOpening`.

The standard in Python is to use underscore seperations for variable and function names.

Assigning Variables to other Variables

You can assign the value of one variable to another:

[2]:
```
var1 = 3
var2 = var1

print('Variable 1 is', var1)
print('Variable 2 is', var2)
```

```
Variable 1 is 3
Variable 2 is 3
```

When you assign assign a variable using another variable, in most cases it is only the value of the variable that is assigned:

[3]:
```
var1 = 3
var2 = var1

print('Variable 1 is', var1)
print('Variable 2 is', var2)

var1 = 2

print('')
print('Variable 1 is', var1)
print('Variable 2 is', var2)
```

```
Variable 1 is 3
Variable 2 is 3

Variable 1 is 2
Variable 2 is 3
```

Notice how, even though we change the value of `var1`, the value of `var2` remains the same.