Chapter_2_Variables_expressions_and_statements

March 6, 2024

One of the most powerful features of a programming language is the ability to manipulate variables. A variable is a name that refers to a value.

[2]:
```
print(type('Hello, World!'))
```

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

[3]:
```
type('Hello, World!')
```

[3]:
```
str
```

[4]:
```
type(17)
```

[4]:
```
int
```

They can contain both letters and numbers, but they can't begin with a number. It is legal to use uppercase letters, but it is conventional to use only lower case for variables names. The underscore character(_) can appear in a name. It is often used in names with multiple words, such as your_name or airspeed_of_unladen_swallow.

[5]:
```
76trombones = 'big parade'
```

```
  File "<ipython-input-5-ee59a172c534>", line 1
    76trombones = 'big parade'
              ^
SyntaxError: invalid syntax
```

[6]:
```
more@ = 1000000
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-6-874d58278958> in <module>
----> 1 get_ipython().run_line_magic('more', '@ = 1000000')

~\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\venv_python_3
 ↪6\lib\site-packages\IPython\core\interactiveshell.py in run_line_magic(self,␣
 ↪magic_name, line, _stack_depth)
   2324                 kwargs['local_ns'] = sys._getframe(stack_depth).f_local
```

```
   2325             with self.builtin_trap:
-> 2326                 result = fn(*args, **kwargs)
   2327             return result
   2328

~\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\venv_python_3
 ↪6\lib\site-packages\decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__doc__ = func.__doc__

~\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\venv_python_3
 ↪6\lib\site-packages\IPython\core\magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

~\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_Raschka\venv_python_3
 ↪6\lib\site-packages\ipykernel\zmqshell.py in less(self, arg_s)
    343             cont = self.shell.pycolorize(openpy.read_py_file(arg_s,␣
 ↪skip_encoding_cookie=False))
    344         else:
--> 345             cont = open(arg_s).read()
    346         page.page(cont)
    347

FileNotFoundError: [Errno 2] No such file or directory: '@ = 1000000'
```

[7]:
```
class = 'Advanced Theoretical Zymurgy'
```

```
  File "<ipython-input-7-73fc4ce1a15a>", line 1
    class = 'Advanced Theoretical Zymurgy'
          ^

SyntaxError: invalid syntax
```

It turns out that class is one of Python's keywords. The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.

Python 3 has these keywords:

False class finally is return
None continue for lambda try

True def from nonlocal while
and del global not with
as elif if or yield
assert else import pass
break except in raise

You don't have to memorize this list. In most development environments, keywords are displayed in a different color; if you try to use one as a variable name, you'll know.

0.1 Setup

http://www.allendowney.com/wp/books/think-python-2e/

For mathematical operators, Python follows mathematical convention. The acronym PEMDAS is a useful way to remember the rules

Parentheses have the highest precedence Exponentiation has the next highest precedence, so 1 + 2**3 is 9, not 27 Multiplication and Division have higher precedence than Addition and Subtraction Operators with the same precedence are evaluated from left to right (except exponentiation).

[8]:
```
#The + operator performs string concatenation, which means it joins the strings␣
 ↪by linking
#them end-to-end. For example:
first = 'throat'
second = 'warbler'
first + second
```

[8]:
```
'throatwarbler'
```

The * operator also works on strings; it performs repetition. For example, 'Spam'*3 is 'SpamSpamSpam'. If one of the values is a string, the other has to be an integer.

This comment is redundant with the code and useless:
v = 5 # assign 5 to v
This comment contains useful information that is not in the code:
v = 5 # velocity in meters/second.
Good variable names can reduce the need for comments, but long names can make complex expressions hard to read, so there is a tradeoff

[ ]:
```

```