Assignment and Substitution

Saturday, May 4, 2019 7:37 PM

The most important programming principle: **substitution**.

If we execute the line of code

```
x = ['some terrible mess']
```

then after this is executed,

- `x` and `['some terrible mess']` can be used interchangeably in subsequent code.
- We can write afterward that `x == ['some terrible mess']`, which we read as "`x` is equal to `['some terrible mess']`"

An expert note:

- `x = something` is a *statement.* This indicates to do something.
- `x == something` is a *conditional.* This checks that something is true.
- After the statement `x = something`, the condition `x == something` is `True`.

Understanding substitution: **`pprint`**

- `pprint(x)` means "pretty-print x".
- We can call this to check on what x's substitution value is.
- For example, if we execute:

  ```
  x = ['hi', 'ho']
  ```

- then after this,

  ```
  from pprint import pprint
  ```

```
   pprint(x)
```
and
```
   pprint(['hi', 'ho'])
```
print exactly the same thing:
```
   ['hi',  'ho']
```
- The `import` only has to be done once. One can call `pprint` several times after one `import.`

An expert note

- The code
```
   from pprint import pprint
```
  makes a function available from module `pprint`, with name `pprint`.
- The full syntax is
```
   from <module> import <name> as <alias>
```
- Thus, you will often see data scientists write this:
```
   from pprint import pprint as pp
```
  after which they can write `pp(x)` rather than `pprint(x)!`

Uses of substitution

- remembering important values for later reuse.
```
   name = 'Alva'
```
- simplifying complex expressions.
```
a = 'apple'
p = 'pear'
fruits = [a, p]
pprint(fruits)
```
prints
```
['apple', 'pear']
```

Data scientists use substitution constantly:

- To determine parameters for use in subsequent computing.
- To save results of intermediate computations.