Python for Astronomers

Script files, if statement, truth, equaltity and identity, tuples, and dictionaries

Exercises

- File /home/rschaaf/test.dat contains two columns with (random) floating point data. Create a new file that has three columns: Columns 1 and 2 are the data from /home/rschaaf/test.dat, column 3 the sum of the numbers in column 1 and 2.

Exercise: Solution

- See script ~rschaaf/solution3.py

Executing scripts

- Run script from Unix shell with

```
> python ~rschaaf/solution3.py
```

- Alternative: Make script executable

```
> cp ~rschaaf/solution3.py sol3
> chmod u+x sol3
> # Add line #!/usr/bin/python
> # at top of sol3
> ./sol3
```

- Or run script from within Python

```
>>> execfile("/home/rschaaf/solution3.py")
```

Docstrings

- Yet another way to run the script:

```
> cp ~rschaaf/solution3.py solution3.py
> python
>>> import solution3 # Will be covered later
>>> # Leading comment now avaiable:
>>> print solution3.__doc__
>>> help(solution3)
>>> <Ctrl>-d
> ls                    # solution3.pyc created
> pydoc solution3       # Runs script
> pydoc -g solution3 # html output
```

Exersise: Style issues

Python Extension Proposals (PEPs) provide much additional information about Python.

- Visit www.python.org/dev/peps/ and browse PEP8 (Style guide) and PEP257 (Docstring conventions)

The if statement

```
>>> if x<0:
...     x = 0

>>> if x<0:
...     print "Oops"
... elif x==0:
...     print "No"
... elif x==1:
...     print "One"
... else:
...     print "Many"
```

Boolean Truth

```
>>> # Comparions yield Boolean values
>>> t = 1==1      # Boolean: True
>>> f = 0==1      # Boolean: False

>>> # Boolean operators:
>>> t and False
>>> True or f
>>> not True

>>> # Precedence: not > and > or:
>>> (not False or True) == (not (False or True))
```

Other Truths

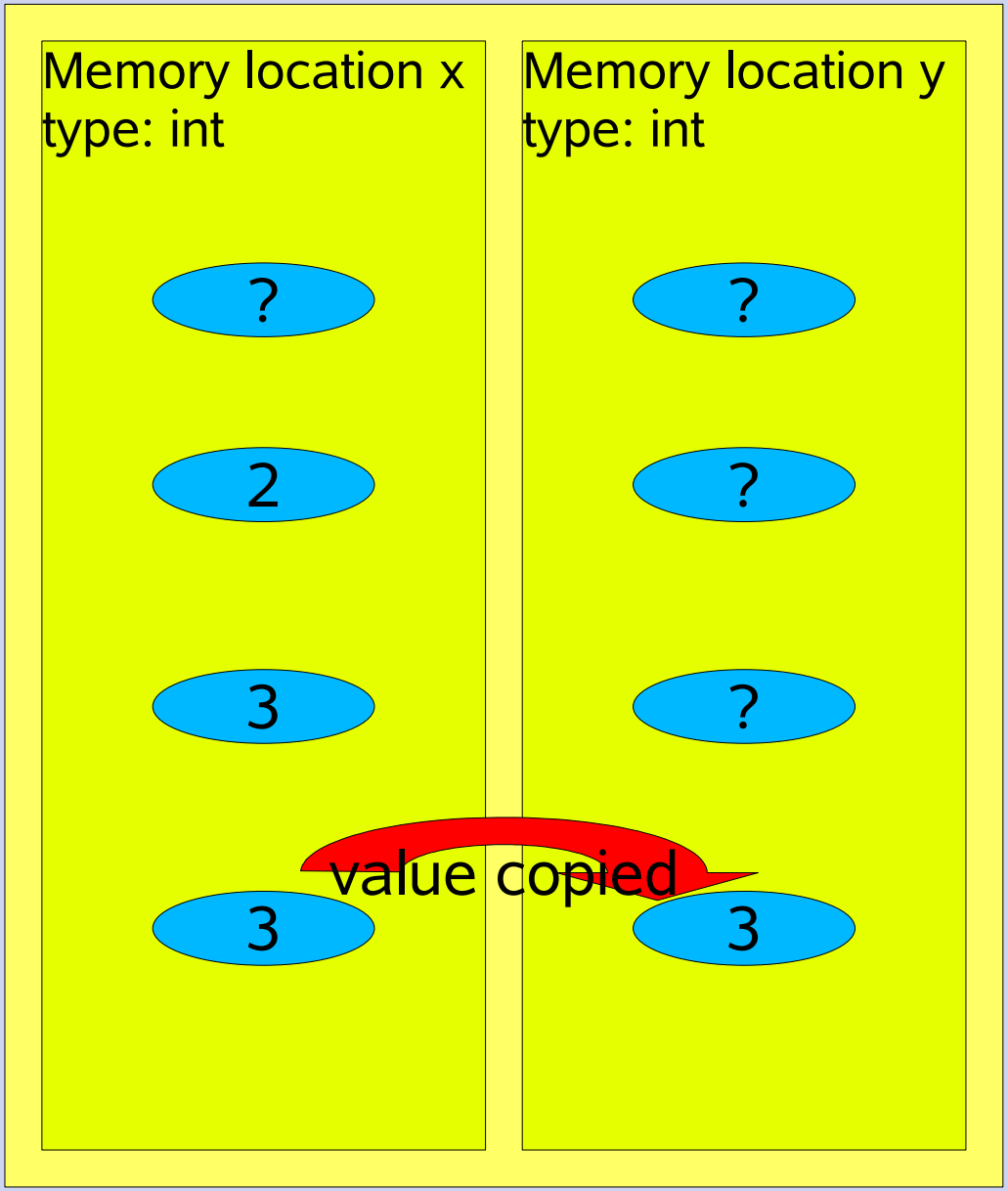
```
>>> # Other datatypes can also express truth:
>>> x = 42
>>> if x:
...   print "true"
... else:
...   print "false"

>>> # Repeat with floats, strings, lists
```

Different truths

```
>>> # Boolean expression can combine
>>> # different datatypes:
>>> 0 or "not empty"
>>> False or "" or [1,2,3] or 42
>>> 42 and [1,2,3]
>>> [1,2,3] and 42

>>> # Boolean expressions are evaluated
>>> # from left to right and return the value
>>> # that determines result
>>> # (Short-circuit evaluation)
```

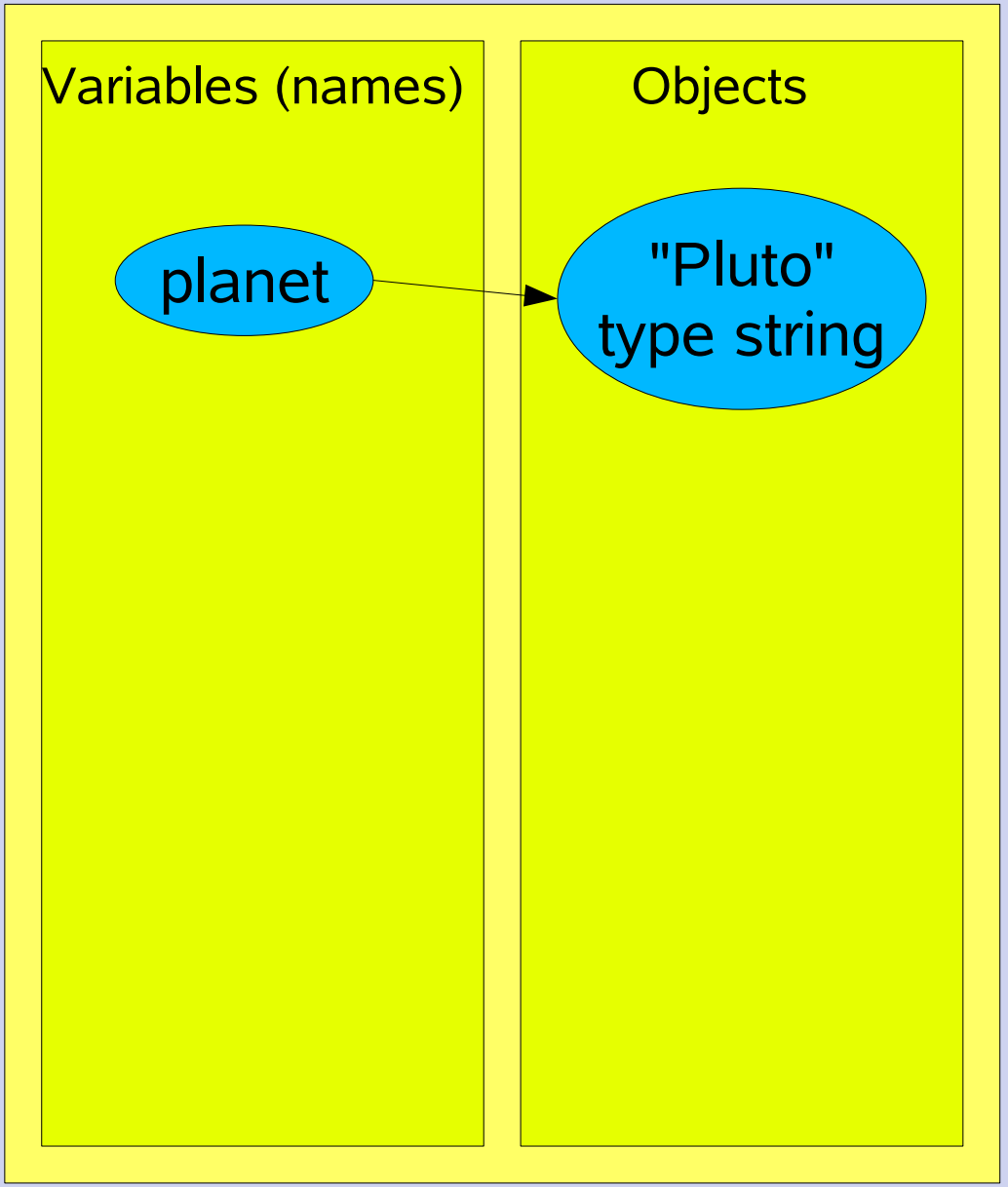
Disgression: Variables in C

```
int x, y;

x = 2;

x = 3;

y = x;
```



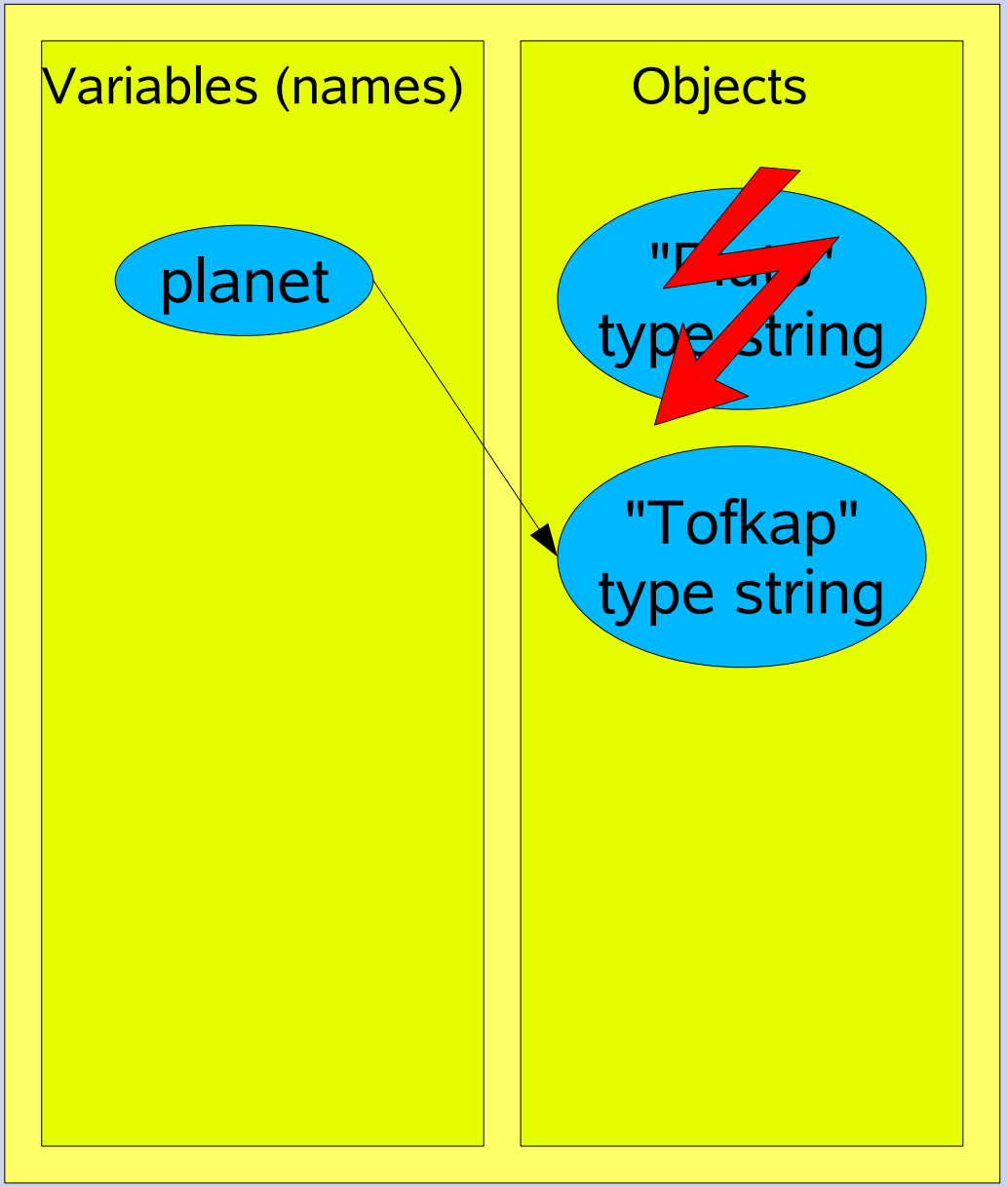
Variables in Python

```
>>> planet = "Pluto"
```



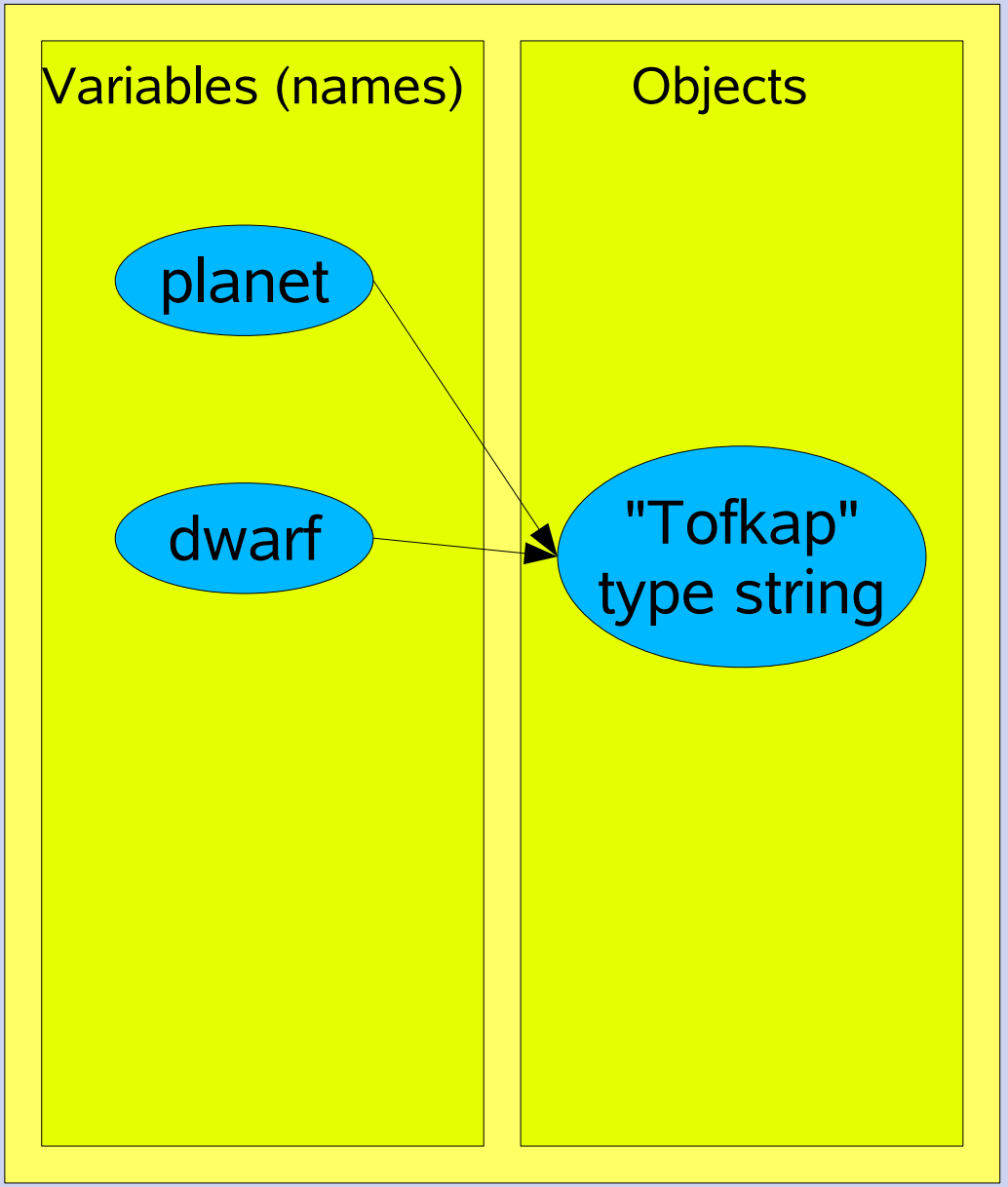
Variables in Python

```
>>> planet = "Pluto"
>>> planet = "Tofkap"
```



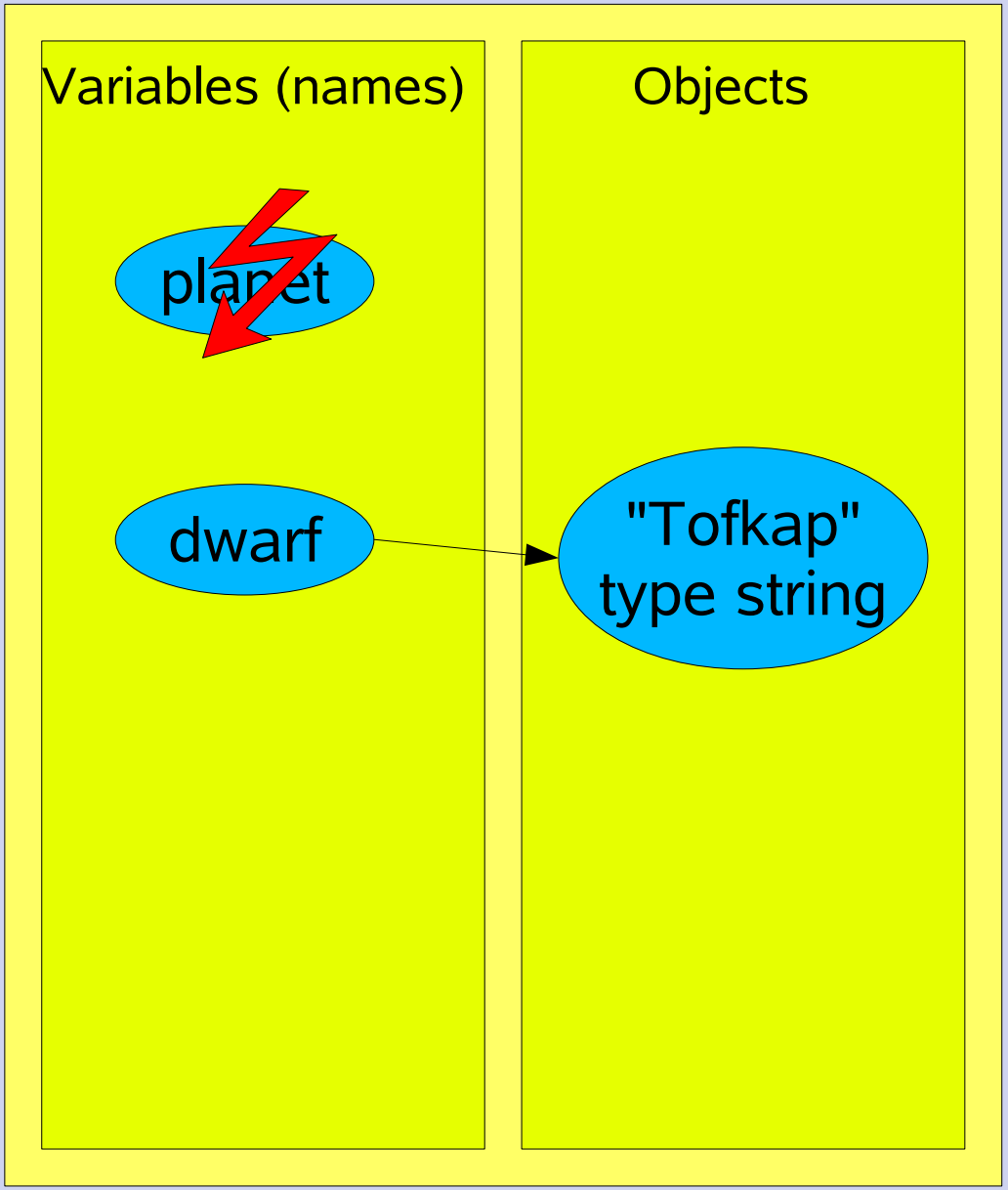
Variables in Python

```
>>> planet = "Pluto"
>>> planet = "Tofkap"
>>> dwarf = planet
```



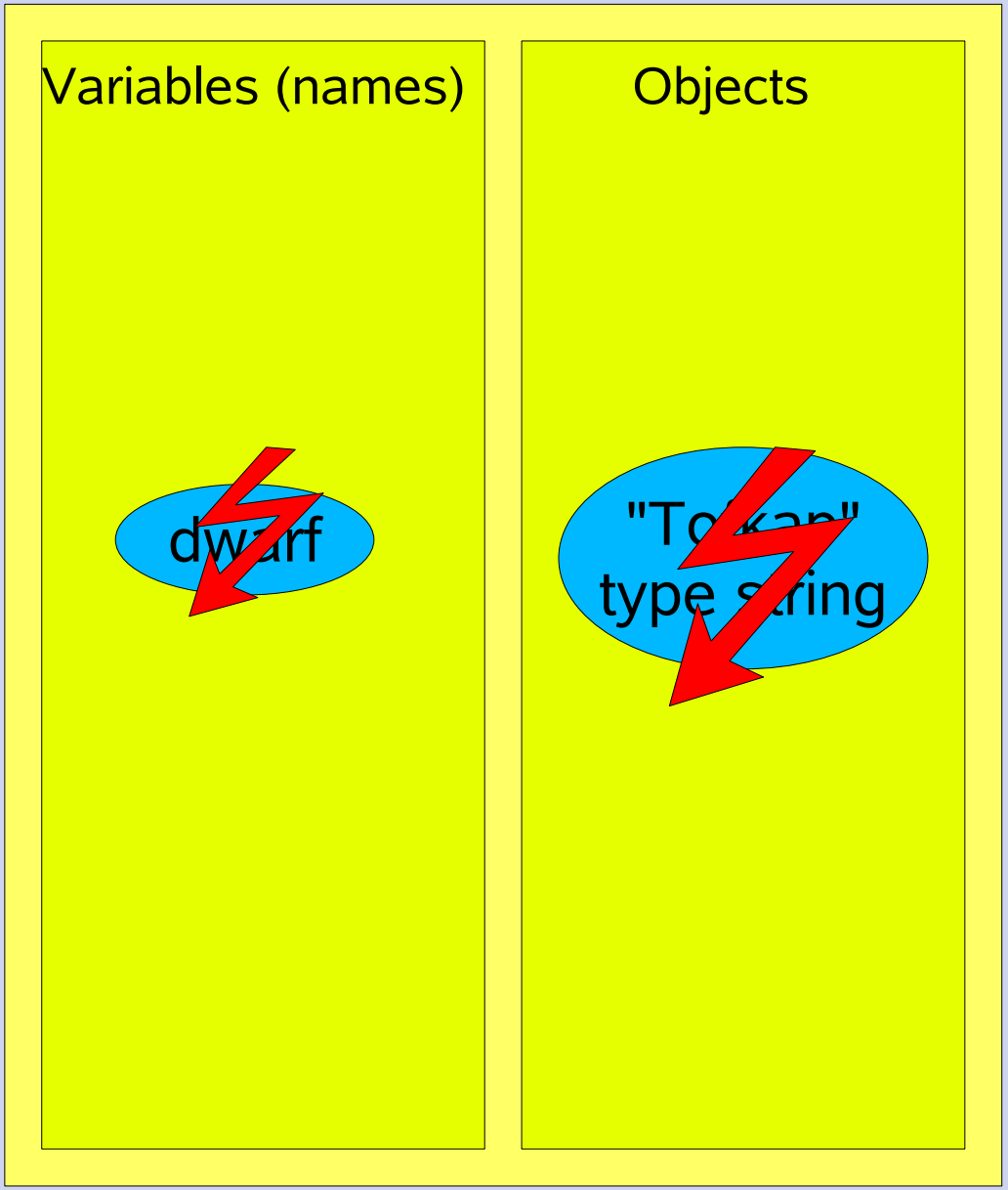
Variables in Python

```
>>> planet = "Pluto"
>>> planet = "Tofkap"
>>> dwarf = planet
>>> del planet
```



Variables in Python

```
>>> planet = "Pluto"
>>> planet = "Tofkap"
>>> dwarf = planet
>>> del planet
>>> del dwarf
```



Equality and Identity

```
>>> l = [1,2,3]
>>> m = [1,2,3]

>>> # Equality (of values) tested with ==
>>> l == m

>>> # Identity (of objects) tested with is
>>> l is m          # l and m different objects

>>> l[0] = 42
>>> print l
>>> print m         # m[0] = ?
```

Equality and Identity

```
>>> l = [1,2,3]
>>> m = l

>>> # Equality (of values) tested with ==
>>> l == m

>>> # Identity (of objects) tested with is
>>> l is m          # l and m same object!

>>> l[0] = 42
>>> print l
>>> print m         # m[0] = ?
```

Exercises: Identity

- Test for identity in the following cases. Do the tests both with the is operator and by changing one of the two objects!

```
>>> l, m = [1,2,3], [1,2,3]
>>> l = m = [1,2,3]
>>> m = l[:]


>>> import numpy
>>> a = numpy.array([1,2,3])
>>> b = a[:]
>>> b = a.copy()
```

Tuples

Sequence types so far:

- Strings: ordered, immutable
- Lists: ordered, mutable

```
>>> # Tuples: ordered, immutable lists
>>> t = (1,2,3)

>>> # Indexing and slicing as with lists:
>>> t[0], t[-1], t[1:]
```

Tuples

```
>>> t = (1,2,3)
>>> t[0] = 42                # Immutable!
>>> t.append(42)             # Immutable!


>>> # But:
>>> t = ([1,2,3],[4,5])
>>> t[0][0] = 42
>>> print t
```

Tuples

```
>>> # Special tuples:
>>> t = ()                  # Empty tuple
>>> t = ("Single",)         # Trailing comma important:
>>> type(t)
>>> t = ("Single")
>>> type(t)


>>> # Parentheses often dropped:
>>> t = 1, 2, 3
```

Tuples

```
>>> # Conversion between tuples and lists:
>>> t = tuple(range(10))
>>> l = list(t)
```

Dictionaries

```
>>> # Dictionaries: unordered, mutable set of
>>> # key:value pairs
>>> d = {'spam': 2, 'eggs': 3}
>>> print d          # Order not preserved

>>> # Use key instead of index:
>>> d['spam']
>>> d['spam'] = 1000000

>>> d['bacon']      # Booo!
```

Dictionaries

```
>>> d = {'spam': 2, 'eggs': 3}

>>> # Either check in advance:
>>> d.has_key('bacon')
>>> # Or use get method:
>>> d.get('bacon', 42)   # Default return value 42

>>> # Adding items:
>>> d['bacon'] = 13

>>> # Removing items:
>>> del d['spam']
```

Dictionaries: Keys and values

```
>>> # Keys are unique for a given dictionary
>>> # Keys may be of any immutable type:
>>> d = {[1,2]: 1}      # Lists are mutable!
>>> d = {(1,2): 1}      # Tuples immutable: ok
>>> # Different keys may have different types:
>>> d['abc'] = 2


>>> # Values may be of any type:
>>> d[42] = ["the answer", [17, 4]]
>>> # Values need not be unique:
>>> d[43] = ["the answer", [17, 4]]
```

Copying dictionaries

```
>>> d1 = {'answer': 42,
...       'food': {'ham': 1, 'spam': 2}}
>>> d2 = d1.copy()
>>> d2['answer'] = 13
>>> d2['food']['ham'] = 0
>>> print d1, d2    # Shallow copy!

>>> import copy
>>> d2 = copy.deepcopy(d1)
>>> d2['answer'] = 'A million'
>>> d2['food']['ham'] = 10
>>> print d1, d2    # Deep copy!
```

Dictionary methods

```
>>> d = {'spam': 2, 'eggs': 3}

>>> len(d)

>>> d.keys(), d.values() # Lists of keys & values

>>> d.clear()    # Returns empty dict. {}
```

Looping over dictionaries

```
>>> d = {'spam': 2, 'eggs': 3, 'bacon': 5}

>>> for k in d:
...   print k, d[k]   # Unsorted

>>> # Clearer equivalent version:
>>> for k in d.keys():
...   print k, d[k]

>>> # Sorted loop:
>>> for k in sorted(d.keys()):
...   print k, d[k]
```

Exercise

The file ~rschaaf/mobydick.txt contains the complete text of Herman Melville's "Moby Dick" (Thanks to the Project Gutenberg www.gutenberg.org)

Find out which the 10 most frequent words in the novel are. Please make sure that your statistics is not affected by interpunction and letter case!