Duck Typing

Duck Typing

```
variable = 3
variable = 'hello'
```

So hasn't variable just changed type ? The answer is a resounding *no*. variable isn't an object at all - it's a name.

In the first statement we create an integer object with the value 3 and bind the name 'variable' to it.

In the second statement we create a new string object with the value hello, and rebind the name 'variable' to it.

If there are no other names bound to the first object then it's reference count will drop to zero and it will be garbage collected.

Duck Typing

"Cuando veo un ave que camina como un pato, nada como un pato y suena como un pato, a esa ave yo la llamo un pato." *Michael Heim. Exploring Indiana Highways*

Python NO usa herencia para definir interfaces

Duck Typing

Python a variable is not an alias for a location in memory. Rather, it is simply a binding to a Python object.

```
string some_guy = "Fred";
// …
```

```
some_guy = 'Fred'
first_names = []
first_names.append(some_guy)

another_list_of_names = first_names
another_list_of_names.append('George') some_guy = 'Bill' print (some_guy, first_names, another_list_of_names)
```

Duck Typing

Python a variable is not an alias for a location in memory. Rather, it is simply a binding to a Python object.

```
some_guy = 'Fred'
first_names = []
first_names.append(some_guy)

another_list_of_names = first_names
another_list_of_names.append('George')
some_guy = 'Bill'
print (some_guy, first_names, another_list_of_names)
```

http://www.jeffknupp.com/blog/2012/11/13/is-python-callbyvalue-or-callbyreference-neither/

Duck Typing

Tthere are actually two kinds of objects in Python. A *mutable*object exhibits time-varying behavior. Changes to a mutable object are visible through all names bound to it. Python's lists are an example of mutable objects. An *immutable* object does not exhibit time-varying behavior.

```
first_names = ['Fred', 'George', 'Bill']
last_names = ['Smith', 'Jones', 'Williams']
name_tuple = (first_names, last_names)

first_names.append('Igor')
```

Duck Typing

```
a = [0,1 2, 3]
print a[0]
0
b = {'a': 0, 'b': 1}
print b['a']
0

a =  [0,1 2, 3]
print list.__getitem__(a, 0)
0
b = {'a': 0, 'b': 1}
print dict._getitem__(b, 'a')
```

Duck Typing

```
if isinstance(object, dict):
    value = object[member]
it is considered more pythonic to do :

try:
     value = object[member]
except TypeError:
    # do something else
```

The principle of duck typing says that you shouldn't care what type of object you have - just whether or not you can do the required action with your object. For this reason the isinstance keyword is frowned upon.

Meta-Programación

Definiciones (Wikipedia)

- **Introspection:** In computing, type introspection is the ability of a program to examine the type or properties of an object at runtime. Some programming languages possess this capability.
- **Reflection:** In computer science, reflection is the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime.
- **Metaprogramming** is the art of writing of computer programs with the ability to treat programs as their data.

Python Implementa estos mecanismos como funcionalidades de primer nivel.

Type

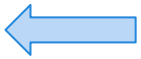
```
>>> type(1)
<type 'int'>
>>> li = []
>>> type(li)
<type 'list'>
>>> import odbchelper
>>> type(odbchelper)
<type 'module'>
>>> import types
>>> type(odbchelper) == types.ModuleType
True
```

Type - Como Clase (Meta-Clase)

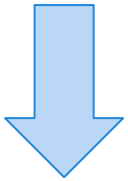
```python
class A(object):

    cls_attr = None

    def __init__(self, attr):
        self.attr = attr
```



Códigos equivalentes



```python
def init(self, attr):
    self.attr = attr

A = type("A", (object,), {"__init__": init, "cls_attr": None})
```

getattr

```
>>> li = ["Larry", "Curly"]
>>> li.pop
<built-in method pop of list object at 010DF884>
>>> getattr(li, "pop")
<built-in method pop of list object at 010DF884>
>>> getattr(li, "append")("Moe")
>>> li
["Larry", "Curly", "Moe"]
>>> getattr({}, "clear")
<built-in method clear of dictionary object at 00F113D4>
>>> getattr((), "pop")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
AttributeError: 'tuple' object has no attribute 'pop'
```

hasattr

```
>>> class Persona(object):

...     def __init__(self, name):
...         self.name = name

>>> p = Persona("tito")
>>> hasattr(p, "name")
True
>>> hasattr(p, "apellido")
False
```

getattr

```
>>> class Persona(object):

...     def __init__(self, name):
...         self.name = name

>>> p = Persona("tito")
>>> setattr(p, "apellido", "puente")
>>> hasattr(p, "apellido")
True
>>> p.apellido
"puente"
```

dir()

```
>>> import os
>>> dir(os)
['EX_CANTCREAT', 'EX_CONFIG', 'EX_DATAERR', 'EX_IOERR', 'EX_NOHOST', 'EX_NOINPUT', 'EX_NOPERM',
'EX_NOUSER', 'EX_OK', 'EX_OSERR', 'EX_OSFILE', 'EX_PROTOCOL', 'EX_SOFTWARE', 'EX_TEMPFAIL',
'EX_UNAVAILABLE', 'EX_USAGE', 'F_OK', 'NGROUPS_MAX', 'O_APPEND', 'O_ASYNC', 'O_CREAT', 'O_DIRECT',
'O_DIRECTORY', 'O_DSYNC', 'O_EXCL', 'O_LARGEFILE', 'O_NDELAY', 'O_NOATIME', 'O_NOCTTY', 'O_NOFOLLOW',
'O_NONBLOCK', 'O_RDONLY', 'O_RDWR', 'O_RSYNC', 'O_SYNC', 'O_TRUNC', 'O_WRONLY', 'P_NOWAIT', 'P_NOWAITO',
'P_WAIT', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'ST_APPEND', 'ST_MANDLOCK', 'ST_NOATIME', 'ST_NODEV'
, 'ST_NODIRATIME', 'ST_NOEXEC', 'ST_NOSUID', 'ST_RDONLY', 'ST_RELATIME', 'ST_SYNCHRONOUS', 'ST_WRITE',
'TMP_MAX', 'UserDict', 'WCONTINUED', 'WCOREDUMP', 'WEXITSTATUS', 'WIFCONTINUED', 'WIFEXITED',
'WIFSIGNALED', 'WIFSTOPPED', 'WNOHANG', 'WSTOPSIG', 'WTERMSIG', 'WUNTRACED', 'W_OK', 'X_OK', '_Environ',
'__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '_copy_reg', '_execvpe',
'_exists', '_exit', '_get_exports_list', '_make_stat_result', '_make_statvfs_result',
'_pickle_stat_result', '_pickle_statvfs_result', '_spawnvef', 'abort', 'access', 'altsep', 'chdir',
'chmod', 'chown', 'chroot', 'close', 'closerange', 'confstr', 'confstr_names', 'ctermid', 'curdir',
'defpath', 'devnull', 'dup', 'dup2', 'environ', 'errno', 'error', 'execl', 'execle', 'execlp', 'execlpe',
'execv', 'execve', 'execvp', 'execvpe', 'extsep', 'fchdir', 'fchmod', 'fchown', 'fdatasync', 'fdopen',
'fork', 'forkpty', 'fpathconf', 'fstat', 'fstatvfs', 'fsync', 'ftruncate', 'getcwd', 'getcwdu', 'getegid',
 'getenv', 'geteuid', 'getgid', 'getgroups', 'getloadavg', 'getlogin', 'getpgid', 'getpgrp', 'getpid',
'getppid', 'getresgid', 'getresuid', 'getsid', 'getuid', 'initgroups', 'isatty', 'kill', 'killpg',
'lchown', 'linesep', 'link', 'listdir', 'lseek', 'lstat', 'major', 'makedev', 'makedirs', 'minor', 'mkdir'
, 'mkfifo', 'mknod', 'name', 'nice', 'open', 'openpty', 'pardir', 'path', 'pathconf', 'pathconf_names',
'pathsep', 'pipe', 'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'readlink', 'remove',
'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'setegid', 'seteuid', 'setgid', 'setgroups', 'setpgid',
 'setpgrp', 'setregid', 'setresgid', 'setresuid', 'setreuid', 'setsid', 'setuid', 'spawnl', 'spawnle',
'spawnlp', 'spawnlpe', 'spawnv', 'spawnve', 'spawnvp', 'spawnvpe', 'stat', 'stat_float_times',
'stat_result', 'statvfs', 'statvfs_result', 'strerror', 'symlink', 'sys', 'sysconf', 'sysconf_names',
'system', 'tcgetpgrp', 'tcsetpgrp', 'tempnam', 'times', 'tmpfile', 'tmpnam', 'ttyname', 'umask', 'uname',
'unlink', 'unsetenv', 'urandom', 'utime', 'wait', 'wait3', 'wait4', 'waitpid', 'walk', 'write']
```

dir()

```
>>> class Persona(object):
...     def __init__(self, nombre):
...         self.nombre = nombre
...

>>> p = Persona("juan")
>>> dir(p)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__hash__',
'__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'nombre']
```

dir()

```
>>> class Persona(object):
...     def __init__(self, nombre):
...         self.nombre = nombre
...

>>> p = Persona("juan")
>>> dir(p)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__', '__hash__',
'__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'nombre']
```

dir(...)

```
dir([object]) -> list of strings

If called without an argument, return the names in the current scope.
Else, return an alphabetized list of names comprising (some of) the attributes
of the given object, and of attributes reachable from it.
If the object supplies a method named __dir__, it will be used; otherwise
the default dir() logic is used and returns:
  for a module object: the module's attributes.
  for a class object:  its attributes, and recursively the attributes
    of its bases.
  for any other object: its attributes, its class's attributes, and
    recursively the attributes of its class's base classes.
```

vars()

```
>>> class Persona(object):
...     def __init__(self, nombre):
...         self.nombre = nombre
...

>>> p = Persona("juan")
>>> vars(p)
{'nombre': 'juan'}
```

```
vars(...)
    vars([object]) -> dictionary

    Without arguments, equivalent to locals().
    With an argument, equivalent to object.__dict__.
```

isinstance()

```
>>> class Persona(object):
...     def __init__(self, nombre):
...         self.nombre = nombre
...
>>> class Empleado(Persona): pass

>>> p = Persona("juan")
>>> e = Empleado("tito")

>>> isinstance(e, Empleado)
True
>>> isinstance(p, Persona)
True
>>> isinstance(e, Persona)
True
>>> isinstance(1, int)
True
>>> isinstance("hola", (int, str))
True
```

Modulo types

```
>>> import types

>>> isinstance(None, types.NoneType)
True

>>> def function(): pass
...
>>> isinstance(function, types.FunctionType)
True


>>> isinstance(range, types.FunctionType)
False
```

Modulo types

```
>>> import types

>>> isinstance(None, types.NoneType)
True

>>> def function(): pass
...
>>> isinstance(function, types.FunctionType)
True


>>> isinstance(range, types.FunctionType)
False
```

```
>>> isinstance(range, types.BuiltinFunctionType)
True
```

callable()

```
>>> def function(): pass
...

>>> callable(function)
True

>>> callable(range)
True

>>> callable(int)
True

>>> callable(Persona)
True

>>> class Persona(object):
...     def metodo(self): pass

>>> callable(Persona.metodo)
True

>>> p = Persona()
>>> callable(p.metodo)
True
```

Esta **desrecomendado** (y no existe en python 3) pero van a ver mucho código con esto

Es mejor usar

```
hasattr(obj, "__call__")
```

O mejor

```
Import types as t
isinstance(obj, (t.FunctionType, t.MethodType,
...))
```

Modulo inspect

- Tiene varias funcionalidades interesantes como **`inspect.isclass()`** o **`inspect.isfunction()`.**
- Hace falta **`inspect.isclass()`** para utilizar con seguridad **`issubclass()`**

```
>>> class Persona(object): pass


>>> def subclase_de_persona(obj):
...     return issubclass(obj, Persona)
...
>>> subclase_de_persona(Empleado)
True
>>> subclase_de_persona(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in subclase_de_persona
TypeError: issubclass() arg 1 must be a class

>>> def subclase_de_persona(obj):
...     return inspect.isclass(obj) and issubclass(obj, Persona)
...
>>> subclase_de_persona(Empleado)
True
>>> subclase_de_persona(1)
```

Finalizando

- Ha más módulos para hacer cosas “locas”
  - **ast** para investigar el árbol abstracto e syntaxis (googlen python hy)
  - **dis** para desensamblar código.

Casi toda librería muy buena en python usa una combinación de lo visto hoy con magic-methods