

Why PYTHON is THE computer language for physicists

A little overview

March 24, 2015

What is PYTHON?

Why use it?

Libraries for physicist

Use examples

Guido Van Rossum:

- Dutch
- Computer scientist
- Benevolent Dictator
For Life

1989: first version coded
during Christmas break



python



- 2.7: since 2010, still supported
- 3.4: since 2014, fixed some of the language flaws. Recommended.

```
user@host:~$ python
Python 2.7.6 (default, Mar 22 2014,
      22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "
    license" for more information.
>>>
```

- Similar to Perl/Bourne again shell (bash)
- No compilation needed
- Easy to test code bits

Everything is an object!

```
class A():  
    pass
```

```
my_object = A  
my_instance = my_object()  
a = 345346345.45753454  
a.as_integer_ratio()
```

When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.¹

¹Heim, Michael (2007). Exploring Indiana Highways. Exploring America's Highway. p. 68. ISBN 978-0-9744358-3-1.

Python is:

- strongly typed

```
>>> a = "something" + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and
      'int' objects
```

- not statically typed

```
>>> a = "something"
>>> a = 23
```


'Natural' language:

```
if my_value in my_list:
    print(my_value)
elif 1 < my_value < 3:
    my_value *= 34
elif my_value and (my_other_value or
something):
    do_something()
else:
    raise Exception("Bad!")
```

Concise:

```
def func(a, b):  
    return a+b  
  
class MyClass(object):  
    def __init__(self):  
        self.attribute = None  
  
    def class_meth(self, a):  
        self.attribute = a  
  
a = MyClass().class_meth(32)
```

Easy iteration:

```
for i in range(32):  
    print i
```

```
a = [float(x) for x in range(23) if x < 32]
```

Dictionaries:

```
a = {"abc": 23, "something": "else",  
      345: "toto"}  
a["test"] = MyObject()  
print a.keys()  
# returns ['abc', 'something', 345, 'test']  
print a.values()  
# returns [23, "else", "toto", <__main__.  
#         MyObject instance at 0x7faf209e5368>]  
print "abc" in a  
# returns True
```

Can contain anything. Keys must be unmutable.

```
def isprime(startnumber):  
    """  
    This function checks if a number is  
    prime or not.  
    """  
    startnumber*=1.0  
    for divisor in range(2,int(startnumber  
        **0.5)+1):  
        if startnumber/divisor==int(  
            startnumber/divisor):  
            return False  
    return True
```

Typical C++ construct:

```
std::map< float, std::list< std::pair < std  
    ::string, double > > > a;  
std::pair< std::string, double > item ("key"  
    , 23.5);  
std::list< std::pair< std::string, double >  
    > my_list (item);  
//be carefull with >> vs > >  
a[21.] = my_list;
```

Equivalent PYTHON code:

```
a = {21.: [("key", 23.5)]}
```

Python dictionaries replace most C/C++
structures

C++ construct:

```
if ( (val > 2 && val < 5) || val2) {  
    std::cout << val << std::endl;  
}  
  
it = map.find("key");  
if (it != map.end() && NULL != it) {  
    std::cout << it->second << std::endl;  
}
```

PYTHON:

```
if 2 < val < 5 or val2:  
    print val  
  
if "key" in map:  
    print map["key"]
```

C++ (in .h):

```
#include "parent.h"

class MyObject: public parent {
public:
    MyObject(int arg) {parent(arg); }
    ~MyObject();

    int my_function(int a);
    void my_other_func();
};
```

in .cpp:

The function definitions such as:

```
int MyObject::my_function(int a) {
    return a + 2;
}
```


PYTHON:

```
from parent import Parent

class MyObject(Parent):
    def __init__(self, arg):
        super(MyObject, self).__init__(self,
                                         arg)

    def my_function(self, a):
        return a

    def my_other_func(self):
        pass
```


