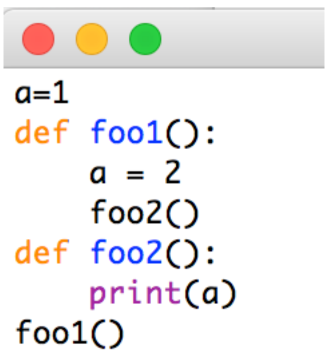
Python

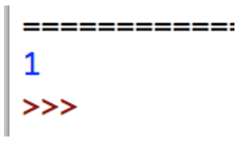
1. Does the language have static scoping, dynamic scoping, both, or other kind of scoping? Explain.

Python use static scoping to confirm this I make an experiment to support this idea here's a result.



```
a=1
def foo1():
    a = 2
    foo2()
def foo2():
    print(a)
foo1()
```

Source Program



```
============
1
>>>
```

Output

It clear that value of "a" was changed before call foo2 but by closet nested scope rules of static scoping the output appears to be the original one.

source: https://hircus.wordpress.com/2005/06/02/static-lexical-vs-dynamic-scoping/

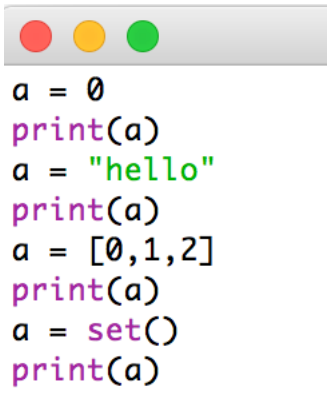
2. Does the language have control flow construct? Explain.

Python have many useful control flow likes if, for, while, break, continues, pass statement and range function. there are more details about it in the source link.

source: https://docs.python.org/2/tutorial/controlflow.html

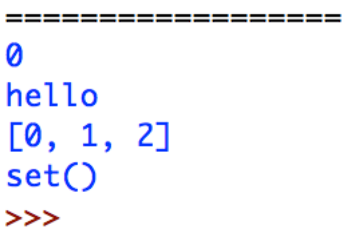
3. Is the language statically typed, dynamically typed, both, or other kind? Explain.

Python is dynamically typed because in python there is no "variable declaration" it only has an "assignment" but "naming" is most suitable word.



```
a = 0
print(a)
a = "hello"
print(a)
a = [0,1,2]
print(a)
a = set()
print(a)
```

Source Program



```
==================
0
hello
[0, 1, 2]
set()
>>>
```

Output

In both pictures above. it's an evidences that python doesn't has a "variable declaration" because it doesn’t output an error like others statically typed language so this can imply that python is dynamically typed language.

source: http://stackoverflow.com/questions/11007627/python-variable-declaration

4. Does the language use the value model, reference model, both, or other kind of model of variables? Explain.

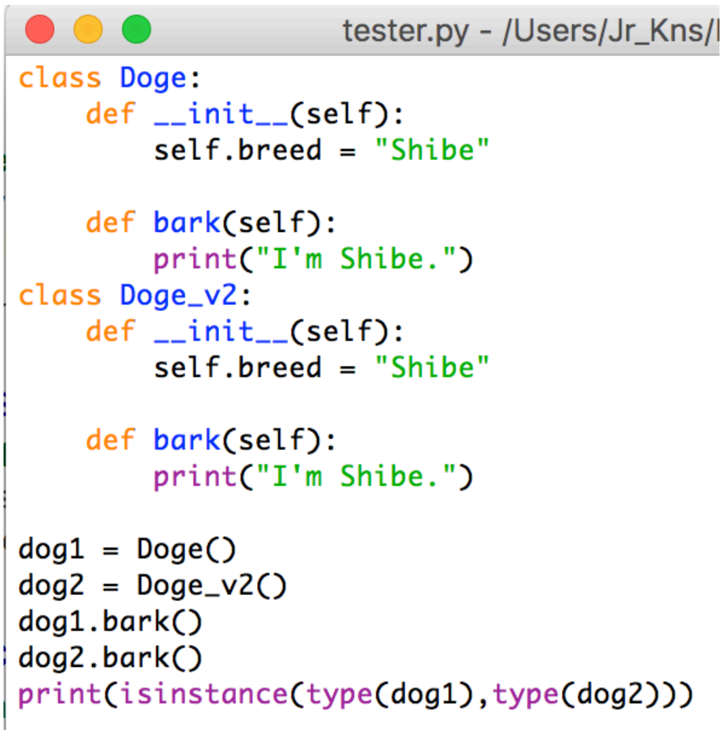
Python use reference model just like in the answer of question 3 python don't has "variable declaration" it’s appear to be more likely "naming to object", so this is the concept of reference model language.

source: same as question 3

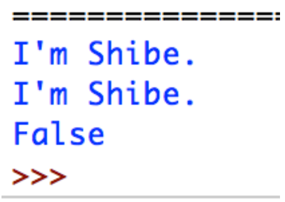
5. Does the language use type equivalence, type compatibility, both, or other kind of typing rules? Explain.

Like others modern language Python use both type compatibility and type equivalence.

First, in type equivalence it's appear to be a name equivalence.


tester.py - /Users/Jr_Kns/

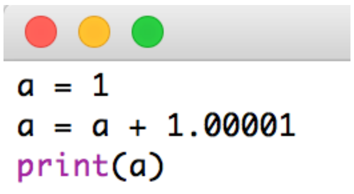
```
class Doge:
    def __init__(self):
        self.breed = "Shibe"

    def bark(self):
        print("I'm Shibe.")
class Doge_v2:
    def __init__(self):
        self.breed = "Shibe"

    def bark(self):
        print("I'm Shibe.")

dog1 = Doge()
dog2 = Doge_v2()
dog1.bark()
dog2.bark()
print(isinstance(type(dog1),type(dog2)))
```



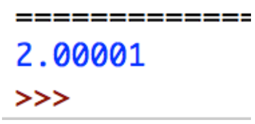
```
================
I'm Shibe.
I'm Shibe.
False
>>>
```

Source Program Output

Second, in type compatibility python is a bit confuse because not only it converts the different one to the type of original one but it's appears to create new object by power of type order.



```
a = 1
a = a + 1.00001
print(a)
```



```
==============
2.00001
>>>
```

Source Program Output

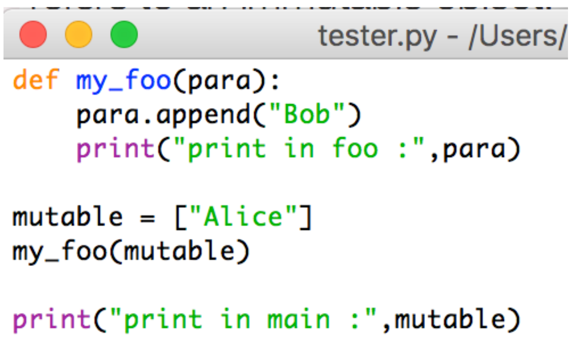
*integer is convert to float at the end.

source: experiment

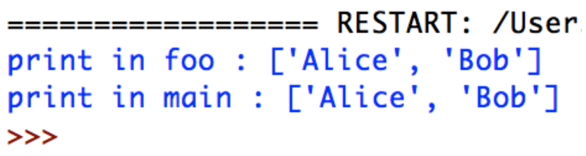
6. Does the language use pass by value, pass by reference, pass by sharing, all of those, or other kind of passing of arguments to subroutines? Explain.

Python is neither pass by value nor pass by reference, it has own passing model called "call-by-object" or "call-by-object-reference".

In python, almost everything is an object and object in python has 2 types (mutable and immutable) so when we pass the mutable object it acts like pass by reference in other language but when we pass the immutable object it act like pass by value so there's a bit of confuse here. So I make an experiment to confirm this.

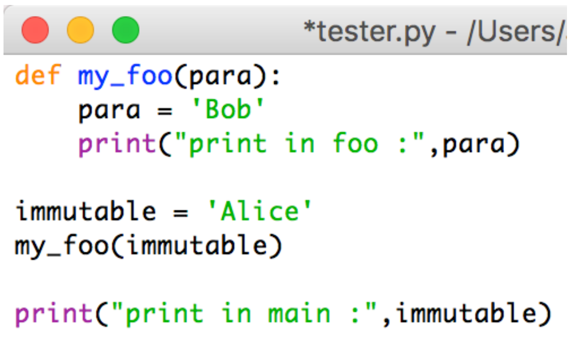


```
def my_foo(para):
    para.append("Bob")
    print("print in foo :",para)

mutable = ["Alice"]
my_foo(mutable)

print("print in main :",mutable)
```

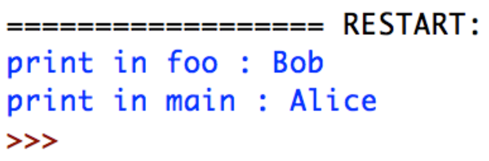


```
================== RESTART: /User
print in foo : ['Alice', 'Bob']
print in main : ['Alice', 'Bob']
>>>
```

Pass a mutable object Output (like pass by ref.)



```
def my_foo(para):
    para = 'Bob'
    print("print in foo :",para)

immutable = 'Alice'
my_foo(immutable)

print("print in main :",immutable)
```



```
================== RESTART:
print in foo : Bob
print in main : Alice
>>>
```

Pass an immutable object Output (like pass by val.)

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