Parameter passing in Python

CSE 216 : Programming Abstractions

Department of Computer Science Stony Brook University

Dr. Ritwik Banerjee

Parameters

- A useful function can usually not operate in complete isolation from its environment.
- **Parameters** provide the mechanism to use objects from “outside” a function, within the function.
- Strictly speaking, **parameters** are inside functions or procedures, while **arguments** are used in procedure calls, *i.e.*, the *values passed to the function at run-time*.
    - But people often use ‘parameter’ and ‘argument’ interchangeably.

Parameter passing

- The classical distinction between two ways of passing parameters has always been in terms of **‘pass by value’** or **‘pass by reference’**.
  - They are also known as “call by value” and “call by reference”, respectively.
- This distinction, however, is becoming increasingly obsolete because the original pass-by-reference technique is no longer favored by newer languages.

Parameter passing

Pass by value

- The argument expression is evaluated, and the result of the evaluation is bound to the corresponding variable in the function.
- That is, if the expression is a variable, its value will be copied to the corresponding parameter.
- The variable in the caller's scope will be unchanged when the function returns.
- Used in C and C++.

Pass by reference

- A function gets a reference to the argument, rather than a copy of its value.
- The value of the variable in the caller's scope can be changed by the function being called.
- Saves time and space in computation.
- C++ supports this. Some languages (*e.g.*, Perl) use this as default.

A Java example

```java
class Main {
  int value = 0;

  static void accept_reference(Main e) {
    e.value++; // will change the referenced object
    e = null; // will only change the parameter
  }

  static void accept_primitive(int v) {
    v++; // will only change the parameter
  }

  public static void main(String... args) { ...
```

A Java example

```
    int value = 0;
    Main ref = new Main(); // reference

    // what we pass is the reference, not the object.
    // we can't pass objects in Java.
    // instead, the reference is copied (pass-reference-by-value)
    accept_reference(ref);
    assert ref != null && ref.value == 1;

    // the primitive int variable is copied
    accept_primitive(value);
    assert value == 0;
  }
}
```

Value Type and Reference Type

Recall that data types can be categorized into **value types** and **reference types**. The names are self-explanatory:	• A value type is an actual value. • A reference type is a reference to another value.
Some languages like Java use a hybrid approach:	• primitives are values types • everything else is a reference type
Python	• *Everything* is a reference type.

Parameter passing in Python

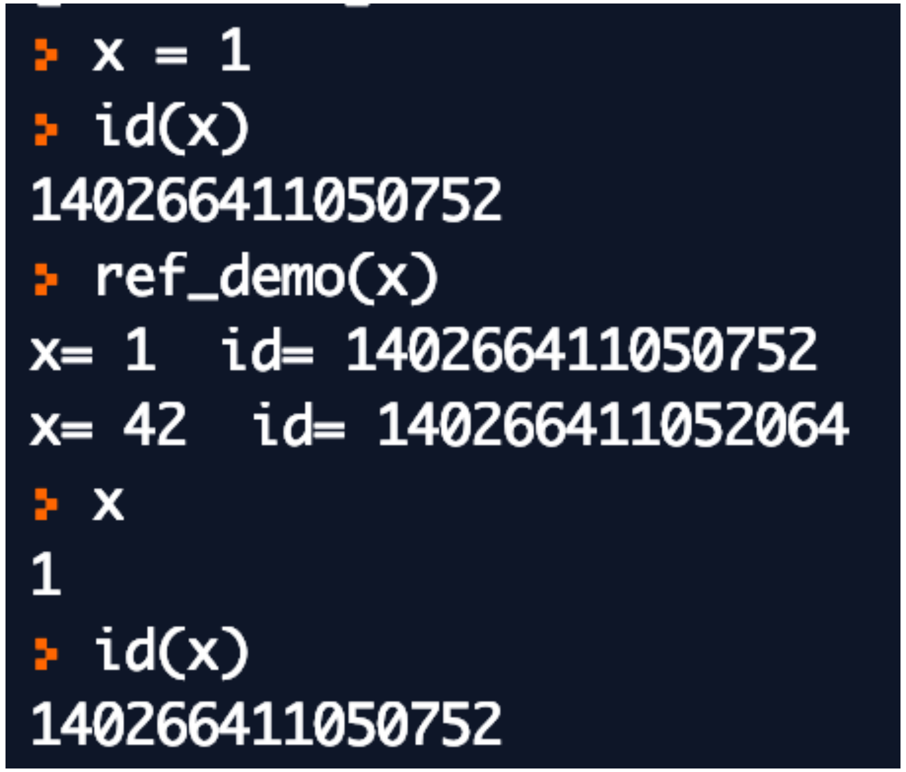
- Python is pass-by-value.

! But remember that since everything is a reference type, <u>the reference is passed by value!</u>.

- When a parameter is passed to a function, the object reference can't be changed from within the called function.
- If the argument itself is mutable, *e.g.*, a list, then there are two scenarios:
  1. The elements of the list can be changed in place. These changes are not changing the reference to the list, and therefore, will be there in the caller's scope.
  2. If a new list is assigned to the same name, the old list will not be affected, *i.e.*, the list in the caller's scope will remain unmodified.

Parameter passing in Python

```python
def ref_demo(x):
    print("x=",x," id=",id(x))
    x=42
    print("x=",x," id=",id(x))
```



```
> x = 1
> id(x)
140266411050752
> ref_demo(x)
x= 1  id= 140266411050752
x= 42  id= 140266411052064
> x
1
> id(x)
140266411050752
```

More Object-Oriented Programming (mostly) in Python

- Magic methods
- Class and Instance attributes
- Properties, getters, setters
- Inheritance
- Operator Overloading
- Class and Type
- Metaclasses and Abstract classes

Magic Methods

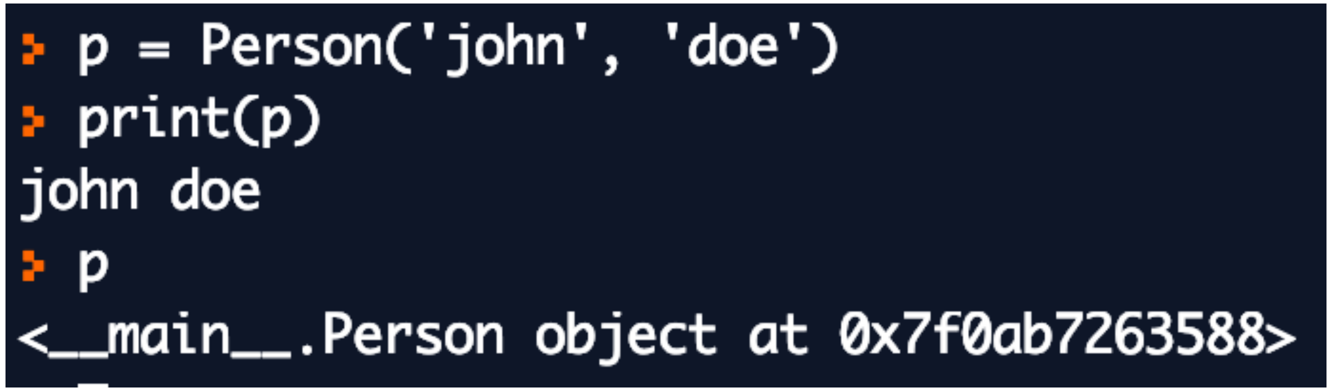
- These are methods with fixed names that serve pre-specified purposes.
  - Python examples: `__init__()`, `__str__()`, `__add__()`.
  - Java examples: `equals()`, `toString()`, `hashCode()`.
- You don't have to call them directly, since they are invoked behind the scene for those pre-specified purposes.
  - Java: the programmer-defined `equals()` method in, say, `MyClass`, gets used to check whether an element already exists in a `Set<MyClass>` when `add()` is invoked.

Magic Methods: constructors and string representations

- The `__init__` method is used to initialize an instance.
- The `__str__` method is used to print an instance.
  - There is a similar method `__repr__`, which is used if `__str__` is not available.

```python
class Person:
    def __init__(self, firstname, lastname=None):
        self.firstname = firstname
        self.lastname = lastname

    def __str__(self):
        return self.firstname + " " + self.lastname
```



```
> p = Person('john', 'doe')
> print(p)
john doe
> p
<__main__.Person object at 0x7f0ab7263588>
```

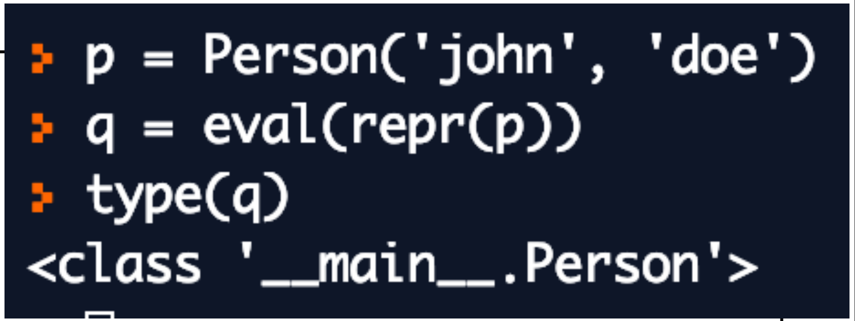
Magic Methods: representation

- An object’s internal representation for the Python interpreter is done using `__repr__`.
- We can get the original object back from the `__repr__` string:

```python
class Person:
    def __init__(self, firstname, lastname=None):
        self.firstname = firstname
        self.lastname = lastname

    def __str__(self):
        return self.firstname + " " + self.lastname

    def __repr__(self):
        return "Person(\"" + self.firstname + "\", \"" + self.lastname + "\")"
```



```
> p = Person('john', 'doe')
> q = eval(repr(p))
> type(q)
<class '__main__.Person'>
```

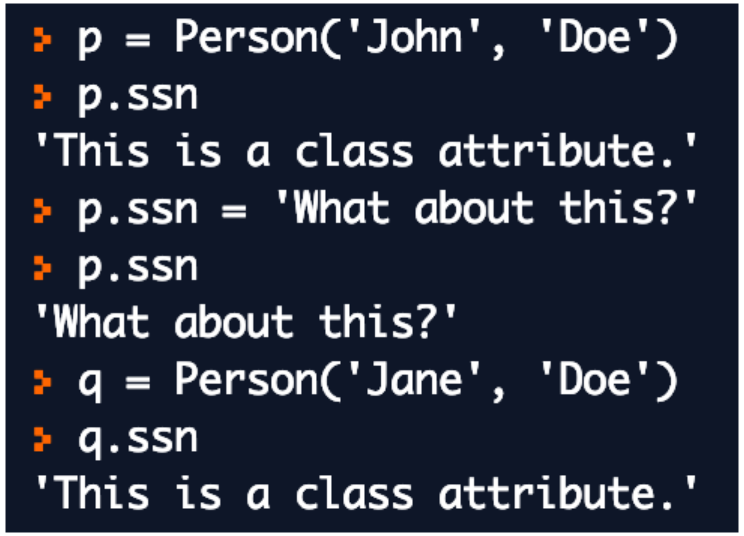
Instance and Class attributes

- As in Java, Python too has attributes whose value depends on the specific instance. These are called **instance attributes**.
- Otherwise, an attribute can have a value that is independent of any specific instance. These are called **class attributes**.

```python
class Person:

    ssn = 'This is a class attribute.'

    ...
```



```
> p = Person('John', 'Doe')
> p.ssn
'This is a class attribute.'
> p.ssn = 'What about this?'
> p.ssn
'What about this?'
> q = Person('Jane', 'Doe')
> q.ssn
'This is a class attribute.'
```

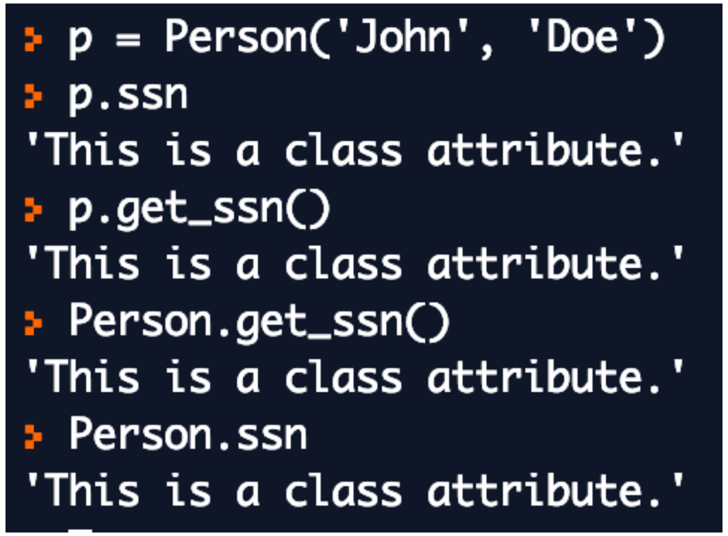
Static methods

- Since class attributes are not instance-specific, instance methods (remember, methods are attributes too) should not be the ones dealing with manipulating them.
- The correct approach is to use **static methods**, where we can call via the class name or via the instance name without the necessity of passing a reference to an instance to it.

```python
class Person:
    ssn = 'This is a class attribute.'

    # __init__ etc.

    @staticmethod
    def get_ssn():
        return Person.ssn
```



```
> p = Person('John', 'Doe')
> p.ssn
'This is a class attribute.'
> p.get_ssn()
'This is a class attribute.'
> Person.get_ssn()
'This is a class attribute.'
> Person.ssn
'This is a class attribute.'
```

Class methods

- Python has a confusing terminology/concept for Java programmers: class methods. These are not tied to instances, but they are not static either!
- In Python, static methods are not bound to a class.
- For class methods, the first parameter is a reference to a class (i.e., a class object).
  - They are often used when we need one static method to call other static methods.
  - Very helpful in various design patterns like the *decorator pattern* and the *factory pattern*. (… CSE 316 stuff that we will not cover).