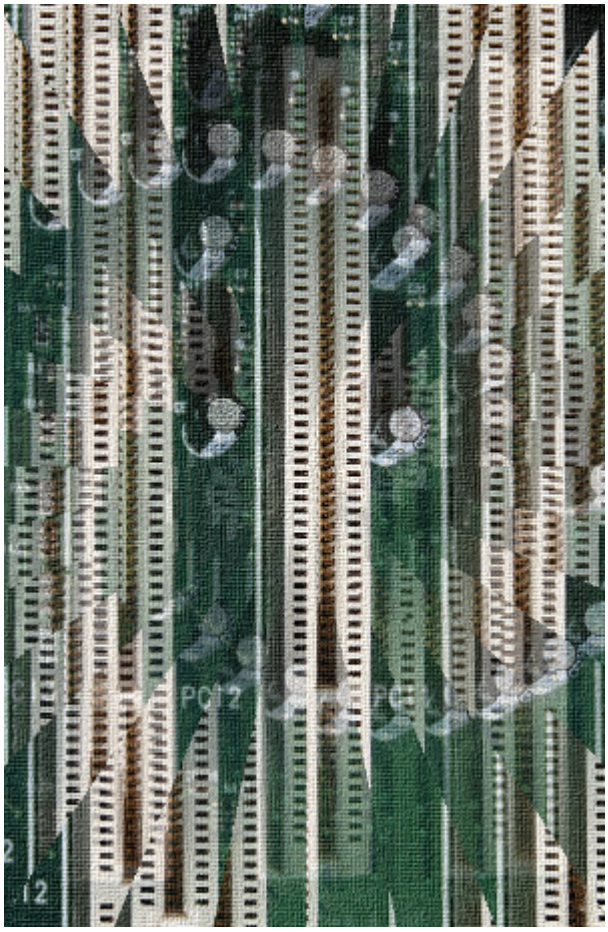
SLOTS

AVOIDING DYNAMICALLY CREATED ATTRIBUTES



The attributes of objects are stored in a dictionary "__dict__". Like any other dictionary, a dictionary used for attribute storage doesn't have a fixed number of elements. In other words, you can add elements to dictionaries after they have been defined, as we have seen in our chapter on dictionaries. This is the reason, why you can dynamically add attributes to objects of classes that we have created so far:

```
>>> class A(object):
...     pass
...
>>> a = A()
>>> a.x = 66
>>> a.y = "dynamically created
attribute"
```

The dictionary containing the attributes of "a" can be accessed like this:

```
>>> a.__dict__
{'y': 'dynamically created
attribute', 'x': 66}
```

You might have wondered that you can dynamically add attributes to the classes, we have defined so far, but that you can't do this with built-in classes like 'int', or 'list':

```
>>> x = 42
>>> x.a = "not possible to do it"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute 'a'
>>>
>>> lst = [34, 999, 1001]
>>> lst.a = "forget it"
Traceback (most recent call last):
```

```
    File "<stdin>", line 1, in <module>
AttributeError: 'list' object has no attribute 'a'
```

Using a dictionary for attribute storage is very convenient, but it can mean a waste of space for objects, which have only a small amount of instance variables. The space consumption can become critical when creating large numbers of instances. Slots are a nice way to work around this space consumption problem. Instead of having a dynamic dict that allows adding attributes to objects dynamically, slots provide a static structure which prohibits additions after the creation of an instance.

When we design a class, we can use slots to prevent the dynamic creation of attributes. To define slots, you have to define a list with the name __slots__. The list has to contain all the attributes, you want to use. We demonstrate this in the following class, in which the slots list contains only the name for an attribute "val".

```
class S(object):

    __slots__ = ['val']

    def __init__(self, v):
        self.val = v


x = S(42)
print(x.val)

x.new = "not possible"
```

If we start this program, we can see, that it is not possible to create dynamically a new attribute. We fail to create an attribute "new":

```
42
Traceback (most recent call last):
  File "slots_ex.py", line 12, in <module>
    x.new = "not possible"
AttributeError: 'S' object has no attribute 'new'
```

We mentioned in the beginning that slots are preventing a waste of space with objects. Since Python 3.3 this advantage is not as impressive any more. With Python 3.3 Key-Sharing Dictionaries are used for the storage of objects. The attributes of the instances are capable of sharing part of their internal storage between each other, i.e. the part which stores the keys and their corresponding hashes. This helps to reduce the memory consumption of programs, which create many instances of non-builtin types.