

## Как питон исполняет программу

- Компиляция в байтокод
- Исполнение байтокода

## Пример программы

```
1  import time
2  from pprint import pprint
3
4  class C(object):
5      pass
6
7  def test_func(x):
8      return x + C()
9
10 pprint(test_func(1))
```

## Ее эквивалент

```
1  time = __import__("time")
2  pprint = __import__("pprint").pprint
3
4  _body = __func__("", compile("return locals()"))
5  C = type("C", _body())
6  del _body
7
8  test_func = __func__("test_func", "return x + C()")
9  pprint(test_func(1))
```

## Байтокод

```
1  import time
2
3  # LOAD_CONST 0 (-1)
4  # LOAD_CONST 1 (None)
5  # IMPORT_NAME 0 (time)
6  # STORE_NAME 0 (time)
7
8
9  def test_func(x):
10     return x + C()
11
12  # func body
13  # LOAD_FAST      0 (x)
14  # LOAD_GLOBAL    0 (C)
15  # CALL_FUNCTION  0
16  # BINARY_ADD
17  # RETURN_VALUE
18
```

```
19      # def statement
20      # LOAD_CONST      5 (<code object .....>)
21      # MAKE_FUNCTION 0
22      # STORE_NAME      4 (test_func)
```

## Исполнение

- Байтокод
- Виртуальная машина (стековая для CPython)
- Место в котором живут все объекты (строки, целые, классы, функции, модули)
- Имена, сгруппированные во множества пространств имен

## Объекты

- С позиции виртуальной машины все объекты эквивалентны
- Все они экземпляры типов, наследующих `PyObject`
- Внутри есть `__class__` - ссылка на объект класса и данные (чаще всего в `__dict__`)
- `PyObject` большая свалка указателей на (виртуальные) функции
- Каждая операция над питон объектом отображается на слот (функция вида `__xxx__`)
- Каждый слот отображается на определенную функцию в `PyObject`

## PyTypeObject

```
1 typedef struct _typeobject {
2     PyObject_VAR_HEAD
3     const char *tp_name; /* For printing ... */
4     /* For allocation */
5     Py_ssize_t tp_basicsize , tp_itemsize;
6
7     destructor tp_dealloc;
8     printfunc tp_print;
9     getattrfunc tp_getattr;
10    // ...
11
12    /* Method suites for standard classes */
13    PyNumberMethods *tp_as_number;
14    PySequenceMethods *tp_as_sequence;
15
16    /* More standard operations ... */
17    // ...
18 } PyTypeObject;
```



## Пространства имен

- Постоянно доступны два пространства имен - локальное и глобальное
- Для них есть ссылки - `locals()` & `globals()`
- Принадлежность к пространству имен определяется на этапе компиляции. Все переменные, которым производится присваивание попадают в `locals()`
- Директивы `globals` & `unlocal`

## Исполнение кода

### Код на питоне

```
1      a.d + C()
```

### Что происходит на самом деле

```
1      a.__getattr__("d").__add__(C.__call__())
```

### Эквивалентный код на C (ceval.c)

```
1      PyObject * c_a = PyTuple_GET_ITEM(co->co_cellvars, 0);
2      PyObject * c_C = PyDict_GetItem(f->f_globals, "C");
3
4      PyObject * c_1 = c_a->ob_type->tp_getattr(a, "d");
5      PyObject * c_2 = c_C->ob_type->tp_call(c_C,
6                                             empty_tuple,
7                                             empty_dict);
8      c_1->ob_type->as_number->nb_add(c_1, c_2);
```