Модули

- python файлы в папке с текущим модулем или в sys.path
- .py => {.pyc, .pyo}
- .pyo = .pyc - asserts
- .pyd / .so
- import mod_name
- import mod_name as new_name
- from mod_name import obj_name, obj_name2
- from mod_name import obj_name as obj_name2
- from mod_name import * все кроме _.* или все из __all__

Модули

```
1   import mod_name
2   # ===
3   mod_name = __import__('mod_name')
4
5   import mod_name as new_name
6   # ===
7   new_name = __import__('mod_name')
8
9   from mod_name import some_name
10  # ===
11  __mod_name = __import__('mod_name')
12  some_name = getattr(__mod_name, 'some_name')
13  del __mod_name
```

Пример модуля - mod.py

```
1       __all__ = ["some_val", "func"]
2
3       some_val = 16
4       internal_data = 1
5
6       def func(x):
7           return x + some_val
```

Использование

```
1       from mod import some_val, func
2       print func(1)
```

Пакеты

```
xml/
    __init__.py
    parsers/
        __init__.py
        expat/
            __init__.py
            bindings.py
        ...
    dom/
        __init__.py
        minidom.py
        ...
    sax/
        __init__.py
        saxutils.py
        ...
    etree/
        __init__.py
        ElementTree.py
        ....
```

Пакеты

- import xml => xml/__init__.py
- import xml.expat => xml/expat/__init__.py
- import xml.expat приводит к появлению имени 'xml' с атрибутом 'expat'
- import xml.etree.ElementTree => xml/expat/ElementTree.py
- Пакет имеет приоритет над модулем при импорте
- __init__.py часто служит для внешнего API пакета

Пакеты - __init__.py

```
1       # __init__.py
2       from some_module1 import api_func_1
3       from some_module2 import api_func_2
4       from some_module3 import ApiClass
5       # ...
```

Циклические импорты

sys.path

- Модифицируемый список всех папок, в которых происходит поиск модулей и пакетов
- Со старта наполняется из :
  - встроенных в питон при компиляции настроек
  - PYTHONPATH
  - реестра
  - xxx.pth
- Может содержать не только директории, при этом python использует пользовательские загрузчики для модулей
- По умолчанию поддерживается импорт из zip архивов

```
1 import sys
2 sys.path.append(SOME_DIR)
3 import module_in_SOME_DIR
```

sys.modules

- Словарь, содержащий отображения имени модуля на объект-модуль.
- import сначала ищет объект по имени в этом словаре, потом вызывает импорт
- mod = reload(mod) перегружает модуль
- при этом все создается новый объект-модуль, но старый не удаляется
- уже созданные объекты остаются связанными со старым модулем

```
1       sys.modules = {
2           # ...
3           'ast': <module 'ast' from '/usr/lib/python2.7/ast.pyc'>,
4            'atexit': <module 'atexit' from '/usr/lib/python2.7/ate
5            'base64': <module 'base64' from '/usr/lib/python2.7/bas
6            'bdb': <module 'bdb' from '/usr/lib/python2.7/bdb.pyc'>
7            'binascii': <module 'binascii' (built-in)>,
8            # ...
9       }
```

```
1       fc = open(module_file_path).read()
2       module_code = compile(fc, module_file, "exec")
3       loc = {}
4       eval(module_code, loc)
5       sys.modules[module_name] = ModuleObject(loc)
6       func = loc['func']
```

Подмена модуля

```
class SomeClass(object):
    def m(self):
        return 1

sys.modules['mod'] = SomeClass()
from mod import m

print m() # 1
```

Специальные атрибуты

- __autor__
- __name__
- __doc__
- __file__
- __name__

```
1       import sys
2
3       def main(argv=None):
4           argv = argv if args is not None else sys.argv
5           ...
6           return code
7
8       if __name__ == "__main__":
9           sys.exit(main())
```

Исполнение модуля

- python -m ....
- python -m timeit -h
- python -m timeit -s "a=b=1" "a+b"
- исполн