with

Типичная работа с файлом

```
>>> file = open(“test”)
>>> try:
>>>     …                                  # do something
>>>     data = file.read()
>>>     …                                  # do something else
>>> finally:
>>>     file.close()
```

```
>>> with open(“test”) as file:
>>>     …                                  # do something
>>>     data = file.read()
>>>     …                                  # do something else
```

Как работает with

```
>>> class Sample(object):
>>>     def __enter__(self):
>>>         print("__enter__")
>>>
>>>     def __exit__(self, type, value, trace):
>>>         print("__exit__")
>>>
>>> with Sample() as sample:
>>>     print("__body__")
__enter__
__body__
__exit__
```

Как работает with

```
>>> class Sample(object):
>>>     def __enter__(self):
>>>         pass                    # Oops...
>>>
>>>     def __exit__(self, type, value, trace):
>>>         pass
>>>
>>>     def do_something(self):
>>>         print(“do something”)


>>> with Sample() as sample:
>>>     sample.do_something()
Traceback (most recent call last):
 File "<stdin>", line 2, in <module>
AttributeError: 'NoneType' object has no attribute
'do_something'
```

Как работает with

```
>>> class Sample(object):
>>>     def __enter__(self):
>>>         print(“enter”)
>>>         return self                    # Why?
>>>
>>>     def __exit__(self, type, value, trace):
>>>         print(“exit”)
>>>
>>>     def do_something(self):
>>>         print(“do something”)

>>> with Sample() as sample:
>>>      sample.do_something()
enter
do something
exit
```

Управление ресурсами

```
>>> class Resource(object):
>>>     def free(self): print("Resource.free()")


>>> class ResourceManager(object):
>>>     def __init__(self, resource):
>>>         super(ResourceManager, self).__init__()
>>>         self.resource = resource
>>>
>>>     def __enter__(self): return self.resource
>>>
>>>     def __exit__(self, type, value, trace): self.resource.free()



>>> with ResourceManager(Resource()) as resource:
>>>     print(resource)
<__main__.Resource object at 0x7f35e570b690>
Resource.free()
```

Метод __exit__

```
>>> class Resource(object):
>>>     def __enter__(self):
>>>         pass
>>>
>>>     def __exit__(self, type, value, trace):
>>>         print("Exception {} of type {}
occured".format(value, type))
>>>

>>> with Resource() as resource:
>>>    raise Exception("Oops...")
Exception Oops... of type <class 'Exception'> occured
Traceback (most recent call last):
 File "<stdin>", line 2, in <module>
Exception: Oops...

>>> with Resource() as resource:
>>>     pass
Exception None of type None occured
```

Метод __exit__

```
>>> class Resource(object):
>>>     def __enter__(self):
>>>         pass
>>>
>>>     def __exit__(self, type, value, trace):
>>>         print(“Exception {} of type {}
occured”.format(value, type))
>>>         return True
>>>

>>> with Resource() as resource:
>>>    raise Exception(“Oops...”)
Exception Oops... of type <class 'Exception'> occured
```

Несколько ресурсов сразу

```
>>> class A:
>>>     def __enter__(self):
>>>         print('A.__enter__')
>>>         return self
>>>
>>>     def __exit__(self, type, value, trace):
>>>         print('A.__exit__')

>>> class B:
>>>     def __enter__(self):
>>>         print('B.__enter__')
>>>         raise Exception('oops')
>>>
>>>     def __exit__(self, type, value, trace):
>>>         print('B.__exit__')
```

Несколько ресурсов сразу

```
>>> with A() as a, B() as b:
>>>     pass
A.__enter__
B.__enter__
A.__exit__
Traceback (most recent call last):
 File "test.py", line 17, in <module>
   with A() as a, B() as b:
 File "test.py", line 12, in __enter__
   raise Exception("oops")
Exception: oops
```

@contextmanager

```
>>> from contextlib import contextmanager
>>>
>>> @contextmanager
>>> def tag(name):
>>>     print("<{}>".format(name))
>>>     yield
>>>     print("</{}>".format(name))
>>>


>>> with tag("h1"):
>>>     print("foo")
<h1>
foo
</h1>
```

Как это работает

```
>>> class GeneratorContextManager
>>>     def __init__(self, func, *args, **kwargs):
>>>         self.gen = func(*args, **kwargs)
>>>     def __enter__(self):
>>>         return next(self.gen)
>>>     def __exit__(self, type, value, trace):
>>>         if value is None:
>>>             try:
>>>                 next(self.gen)
>>>             except StopIteration:
>>>                 return
>>>         else:
>>>             try:
>>>                 self.gen.throw(type, value, trace)
>>>             except StopIteration:
>>>                 return True

>>> def contextmanager(func):
>>>     def helper(*args, **kwargs):
>>>         return GeneratorContextManager(func, *args, **kwargs)
>>>     return helper
```

contextlib

```
>>> from contextlib import closing, ExitStack
>>> from urllib.request import urlopen


>>> with closing(urlopen('http://www.python.org'))
as page:
>>>     … # do somethin
>>> # page.close() called
```