Замыкания

Основы Python

План урока

- Поговорим о замыканиях

ЗАМЫКАНИЯ

это очень серьезный инструмент, который широко используется в программировании

Замыкания работают не только в Python, но и во многих других высокоуровневых языках

ФУНКЦИИ В PYTHON

можно определять где угодно
(в любом блоке кода)

Функции

ooo | код

```
for i in range(5):

    def func():
        print(f'hello, {i}')
    func()
```

Определяем функцию “func”

В функции печатается слово “hello” и номер итерации

ooo | вывод

```
>>> hello, 0
>>> hello, 1
>>> hello, 2
>>> hello, 3
>>> hello, 4
```

Функции

ooo | код

```
for i in range(5):

    def func():
        print(f'hello, {i}')
    func()
```

Номер итерации - это одно из значений “range”, то есть от 0 до 4

ooo | вывод

```
>>> hello, 0
>>> hello, 1
>>> hello, 2
>>> hello, 3
>>> hello, 4
```

Функции

ooo код

```
for i in range(5):

    def func():
        print(f'hello, {i}')
    func()
```

Вызывается в текущем цикле, сразу после определения функции

ooo вывод

```
>>> hello, 0
>>> hello, 1
>>> hello, 2
>>> hello, 3
>>> hello, 4
```

Функции

ooo | код

```
for i in range(5):

    def func():
        print(f'hello, {i}')
    func()
```

ooo | вывод

```
>>> hello, 0
>>> hello, 1
>>> hello, 2
>>> hello, 3
>>> hello, 4
```

Вывели пять строчек “hello” с соответствующими числами

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner


f = outer()
f()
f()
```

Функция “outer”

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner

f = outer()
f()
f()
```

Внутри нее мы определяем функцию “inner”

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner

f = outer()
f()
f()
```

Перед определением функции “inner” в функции “outer” мы печатаем сообщение ”outer start”

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner

f = outer()
f()
f()
```

В функции “inner” печатаем два сообщения: ”inner start” и ”inner end”

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner


f = outer()
f()
f()
```

Печатаем сообщение “outer end”

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner


f = outer()
f()
f()
```

Возвращаем “inner”

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner


f = outer()
f()
f()
```

Присваиваем переменной “f” значение, возвращаемое функцией “outer”

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner


f = outer()
f()
f()
```

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Печатаются сообщения "outer start" и "outer end"

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner

f = outer()
f()
f()
```

Вызываем "f" как функцию

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Пример

ooo | код

```
def outer():
    print('outer start')

    def inner():
        print('inner start')
        print('inner end')

    print('outer end')
    return inner

f = outer()
f()
f()
```

ooo | вывод

```
>>> outer start
>>> outer end
>>> inner start
>>> inner end
>>> inner start
>>> inner end
```

Видим сообщения: ”inner start” и ”inner end” в каждом из случаев

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Определяем функцию “my_counter”

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Вводим переменную “cnt“

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Определяем функцию “increment“

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Увеличиваем “cnt“ на 1

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Увеличиваем “cnt“ на 1

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Возвращаем это значение

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Значение “nonlocal cnt“

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Вызываем функцию “my_counter”, присваивая это значение переменной ”f”

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Вызываем дважды

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Печатаем 1 и 2

Нелокальный контекст

код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Присваиваем значение переменной “my_counter”

вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

Снова вызываем получившуюся функцию

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Нелокальный контекст

ooo | код

```
def my_counter():
    cnt = 0

    def increment():
        nonlocal cnt
        cnt += 1
        return cnt

    return increment

f = my_counter()

print(f())
print(f())

f = my_counter()

print(f())
```

ooo | вывод

```
>>> 1
>>> 2
>>> 1
```

Получаем снова значение 1

НЕЛОКАЛЬНЫЙ КОНТЕКСТ

это набор объектов, определенных в области нелокальной по отношению к функции

ЗАМЫКАНИЯ

это особый вид функции, которая определена в теле другой функции

Замыкания позволяют нам создавать и использовать функции, которые могут сохранять свое состояние из предыдущих вызовов

Пример

КОД

```
def my_counter_with_msg():
    cnt = 0
    msg = 'hi'

    def increment():
        nonlocal cnt
        cnt += 1
        print(msg)
        return cnt

    return increment


f = my_counter_with_msg()

print(f())
print(f())

f = my_counter_with_msg()

print(f())
```

ВЫВОД

```
>>> hi
>>> 1
>>> hi
>>> 2
>>> hi
>>> 1
```

Объявляем переменную “msg“ с текстом “hi“

Пример

ooo | КОД

```
def my_counter_with_msg():
    cnt = 0
    msg = 'hi'

    def increment():
        nonlocal cnt
        cnt += 1
        print(msg)
        return cnt

    return increment


f = my_counter_with_msg()

print(f())
print(f())

f = my_counter_with_msg()

print(f())
```

ooo | ВЫВОД

```
>>> hi
>>> 1
>>> hi
>>> 2
>>> hi
>>> 1
```

При каждом вызове функции “f” печатается сообщение “hi“

Пример

КОД

```
def my_counter_with_msg_name(name):
    cnt = 0
    msg = 'hi'

    def increment():
        nonlocal cnt
        cnt += 1
        print(f'{msg}, {name}')
        return cnt

    return increment

f = my_counter_with_msg_name("Peter")

print(f())
print(f())

f = my_counter_with_msg_name("Steven")

print(f())
```

ВЫВОД

```
>>> hi, Peter
>>> 1
>>> hi, Peter
>>> 2
>>> hi, Steven
>>> 1
```

Добавляем аргумент "name" и используем его в "increment"

Пример

ooo | КОД

```
def my_counter_with_msg_name(name):
    cnt = 0
    msg = 'hi'

    def increment():
        nonlocal cnt
        cnt += 1
        print(f'{msg}, {name}')
        return cnt

    return increment

f = my_counter_with_msg_name("Peter")

print(f())
print(f())

f = my_counter_with_msg_name("Steven")

print(f())
```

ooo | ВЫВОД

```
>>> hi, Peter
>>> 1
>>> hi, Peter
>>> 2
>>> hi, Steven
>>> 1
```

Присваиваем значение функции с параметром "Peter"

Пример

КОД

```
def my_counter_with_msg_name(name):
    cnt = 0
    msg = 'hi'

    def increment():
        nonlocal cnt
        cnt += 1
        print(f'{msg}, {name}')
        return cnt

    return increment

f = my_counter_with_msg_name("Peter")

print(f())
print(f())

f = my_counter_with_msg_name("Steven")

print(f())
```

ВЫВОД

```
>>> hi, Peter
>>> 1
>>> hi, Peter
>>> 2
>>> hi, Steven
>>> 1
```

Получаем "hi, Peter 1" и "hi, Peter 2"

Пример

КОД

```
def my_counter_with_msg_name(name):
    cnt = 0
    msg = 'hi'

    def increment():
        nonlocal cnt
        cnt += 1
        print(f'{msg}, {name}')
        return cnt

    return increment

f = my_counter_with_msg_name("Peter")

print(f())
print(f())

f = my_counter_with_msg_name("Steven")

print(f())
```

ВЫВОД

```
>>> hi, Peter
>>> 1
>>> hi, Peter
>>> 2
>>> hi, Steven
>>> 1
```

Присваиваем значение функции с параметром "Steven"

Пример

```
def my_counter_with_msg_name(name):
    cnt = 0
    msg = 'hi'

    def increment():
        nonlocal cnt
        cnt += 1
        print(f'{msg}, {name}')
        return cnt

    return increment

f = my_counter_with_msg_name("Peter")

print(f())
print(f())

f = my_counter_with_msg_name("Steven")

print(f())
```

ooo | ВЫВОД

```
>>> hi, Peter
>>> 1
>>> hi, Peter
>>> 2
>>> hi, Steven
>>> 1
```

Печатаем "hi, Steven 1"