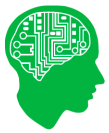

SkillFactory

Профессия Data Scientist

Модуль Функциональное программирование

Простейшая функция

```
def simple_func():
    print("Hello World")

simple_func()  # "Hello World"
```

Функция с обязательными аргументами

```
def func_with_arg(name):
    print(f"Hello {name}")

func_with_arg("username")
# "Hello username"
```

Функция с аргументами по умолчанию

```
def func_default_arg(name="World"):
    print(f"Hello {name}")


func_default_arg()  # "Hello World"
func_default_arg("username")
# "Hello username"
```

Функция возвращающая результат

```
def func_return(name):
    return f"Hello {name}"

msg = func_return("username")
print(msg)  # "Hello username"
```

Функции генераторы

`yield x`	Возвращает значение x, приостанавливает и сохраняет внутреннее состояние
`iter(obj)`	Получить итератор от итерируемого объекта
`next(function)`	Возобновляет выполнение (или запускает генератор в первый раз)
Функции генератор работаю до тех пока не закончатся все значения или не будет вызвано исключение StopIteration	

Замыкание функций

```
def make_adder(x):
    def adder(n):
        return x + n # захват
переменной "x" из nonlocal области
    return adder  # возвращение
функции в качестве результата
```

Args и kwargs

```
def my_function(*args, **kwargs):
    pass
```

Декораторы. “Синтаксический сахар”

```
@my_decorator
def my_function():
    pass
```

Декораторы. Шаблон

```
def my_decorator(fn):
    def wrapper():
        fn()
    return wrapper  # возвращается
задекорированная функция, которая
заменяет исходную

# выведем незадерорированную функцию
def my_function():
    pass
print(my_function)  # <function
my_function at 0x7f938401ba60>

# Декорируем функцию
@my_decorator
def my_function():
    pass

№ Вызываем задекорированную функцию
my_function()
```