Университет Сириус
Колледж

Основы программирования
Лекция 4.

Функции. Пространства имён и области видимости. Замыкания. Генераторы. Декораторы. Обработка ошибок.

Университет Сириус Колледж

Функции

Функции в Python определяются с помощью инструкции def, которое вводит определение функции. За ним должно следовать имя функции и заключенный в скобки список формальных параметров/аргументов. Операторы, которые формируют тело функции, начинаются со следующей строки и должны иметь отступ.cd

```
1  def func_name(param):
2      pass
```

- `func_name` - идентификатор, то есть переменная, которая при выполнении инструкции def связывается со значением в виде объекта функции.
- `param` - это необязательный список формальных параметров/аргументов, которые связываются со значениями, предоставляемыми при вызове функции.

Университет Сириус Колледж

Функции

Функция - объект, такой же как и прочие объекты в Python.

```
1  def hello(name):
2      return f'Hello {name}.'
3
4  say = hello
```

Университет Сириус Колледж

Функции

Функция - объект, такой же как и прочие объекты в Python.

```
1  def hello(name):
2      return f'Hello {name}.'
3
4  say = hello
5  say('World')
6  # Hello World.
```

Функции

Функция - объект, такой же как и прочие объекты в Python.

```
1  def hello(name):
2      return f'Hello {name}.'
3
4  say = hello
5  del hello
6  hello('World')
7  # Traceback (most recent call last):
8  #   File "<stdin>", line 1, in <module>
9  # NameError: name 'hello' is not defined
```

Университет Сириус Колледж

Функции

Функция - объект, такой же как и прочие объекты в Python.

```
1  def hello(name):
2      return f'Hello {name}.'
3
4  say = hello
5  del hello
6  hello('World')
7  # Traceback (most recent call last):
8  #   File "<stdin>", line 1, in <module>
9  # NameError: name 'hello' is not defined
10 say('World')
11 # 'Hello World.'
```

Университет Сириус Колледж

Функции

Функция - объект, такой же как и прочие объекты в Python.

```python
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

a, b = 4, 5
(subtract if a > b else add)(a, b)
# 9
```

Университет Сириус Колледж

Атрибуты функций

У объектов функций есть специальный атрибут `__dict__`. Это словарь атрибутов функции. В него можно устанавливать и получать какие-то значения с помощью точечной нотации.

```
1  def func():
2      func.a = 10
3
4  func.__dict__
5  # {}
```

Университет Сириус Колледж

Атрибуты функций

У объектов функций есть специальный атрибут `__dict__`. Это словарь атрибутов функции. В него можно устанавливать и получать какие-то значения с помощью точечной нотации.

```
1  def func():
2      func.a = 10
3
4  func()
5  func.__dict__
6  # {'a': 10}
7  func.a
8  # 10
```

Университет Сириус Колледж

Атрибуты функций

У объектов функций есть специальный атрибут `__dict__`. Это словарь атрибутов функции. В него можно устанавливать и получать какие-то значения с помощью точечной нотации.

```
1  def func():
2      func.a = 10
3
4  func.a = 25
5  func.a
6  # 25
```

Университет Сириус Колледж

Атрибуты функций

У объектов функций есть специальный атрибут `__dict__`. Это словарь атрибутов функции. В него можно устанавливать и получать какие-то значения с помощью точечной нотации.

```
1  def func():
2      func.a = 10
3
4  func.x = 6
5  func.x
6  # 6
```

Университет Сириус Колледж

Атрибуты функций

У объектов функций есть специальный атрибут `__dict__`. Это словарь атрибутов функции. В него можно устанавливать и получать какие-то значения с помощью точечной нотации.

```
1  def func():
2      func.a = 10
3
4  func.list = []
5  func.list.append(10)
6  func.list.append(1)
7  func.list.append(5)
8  func.list
9  # [10, 1, 5]
10 func.__dict__
11 # {'a': 25, 'x': 6, 'list': [10, 1, 5]}
```

Университет Сириус Колледж

Атрибуты функций

Словарь атрибутов может быть использован для кеширования промежуточных значений декоратора или для кэширования уже вычисленных значений функции. Например это может быть атрибут функции `func.cash`, который будет хранить словарь, у которого в качестве ключа будет кортеж входных параметров функции, а значение словаря - возвращаемый результат функции.

Атрибуты иногда используются как статические переменные для функции.

Университет Сириус Колледж

Вложенные функции

```
1  def parent():
2      print('⇒ parent')
3      def child():
4          print("⇒ I'm child function")
5      child()
6
7  parent()
8  # ⇒ parent
9  # ⇒ I'm child function
```

Важно: *внутренние функции не определены до тех пор, пока не будет вызвана родительская функция. Они локально ограничены родительской функцией* `parent()`*. Они существуют только внутри функции* `parent()` *как локальные переменные.*

Университет Сириус Колледж

Вложенные функции

```python
def talk(n):
    def hello(name):
        return f'Привет {name}.'
    def goodbye(name):
        return f'Пока {name}.'
    if n > 0:
        return hello
    else:
        return goodbye
```

Функции могут не только принимать поведение через аргументы, но и могут возвращать поведение!

Университет Сириус Колледж

Пространства имен, области видимости

- **Пространство имён** - это раздел, внутри которого имя уникально и не связано с такими же именами в других пространствах имён.
- Каждая функция определяет собственное пространство имён.
- Если вы определили переменную x в основной программе, а в функции также определили переменную x, они будут ссылаться на разные значения.
- В основной программе определяется **глобальное пространство имён**, поэтому переменные, находящиеся в нём, являются глобальными.

Университет Сириус Колледж

Области видимости

```
1  from math import ceil # область встроенных имен
2
3  def sum_func():
4      # локальная область видимости функции sum_func()
5      b = a * 10 # 'b'
6      # 'b' - локальная переменная функции sum_func()
7      # НЕ доступна для ЧТЕНИЯ в глобальной области
8      # Доступна для ЧТЕНИЯ в области видимости вложенной функции nested()
9      # НЕ доступна для ИЗМЕНЕНИЯ в области видимости вложенной функции nested()
10     # Здесь 'a' называется свободной переменной
11
12     def nested():
13         # локальная область видимости вложенной функции nested()
14         z = b / 5 * a
```

Университет Сириус Колледж

Области видимости

```python
from math import ceil # область встроенных имен

def sum_func():
    # локальная область видимости функции sum_func()
    b = a * 10 # 'b'
    # 'b' - локальная переменная функции sum_func()
    # НЕ доступна для ЧТЕНИЯ в глобальной области
    # Доступна для ЧТЕНИЯ в области видимости вложенной функции nested()
    # НЕ доступна для ИЗМЕНЕНИЯ в области видимости вложенной функции nested()
    # Здесь 'a' называется свободной переменной

    def nested():
        # локальная область видимости вложенной функции nested()
        z = b / 5 * a
```

Университет Сириус Колледж

Области видимости

```python
from math import ceil # область встроенных имен

def sum_func():
    # локальная область видимости функции sum_func()
    b = a * 10 # 'b'
    # 'b' - локальная переменная функции sum_func()
    # НЕ доступна для ЧТЕНИЯ в глобальной области
    # Доступна для ЧТЕНИЯ в области видимости вложенной функции nested()
    # НЕ доступна для ИЗМЕНЕНИЯ в области видимости вложенной функции nested()
    # Здесь 'a' называется свободной переменной

    def nested():
        # локальная область видимости вложенной функции nested()
        z = b / 5 * a
```

Университет Сириус Колледж

Области видимости

```
1  from math import ceil # область встроенных имен
2
3  def sum_func():
4      # локальная область видимости функции sum_func()
5      b = a * 10 # 'b'
6      # 'b' - локальная переменная функции sum_func()
7      # НЕ доступна для ЧТЕНИЯ в глобальной области
8      # Доступна для ЧТЕНИЯ в области видимости вложенной функции nested()
9      # НЕ доступна для ИЗМЕНЕНИЯ в области видимости вложенной функции nested()
10     # Здесь 'a' называется свободной переменной
11
12     def nested():
13         # локальная область видимости вложенной функции nested()
14         z = b / 5 * a
```

Университет Сириус Колледж

Области видимости

```
23
24
25 # Глобальная область видимости
26
27 a = 10
28 # 'a' - глобальная переменная
29 # Доступна для ЧТЕНИЯ в области видимости функции sum_func()
30 # Доступна для ЧТЕНИЯ в области видимости вложенной функции nested()
31 # НЕ доступна для ИЗМЕНЕНИЯ в области видимости функции sum_func()
32 # НЕ доступна для ИЗМЕНЕНИЯ в области видимости вложенной функции nested()
33
34 print(sum_func())
35 #  напечатает
36 200.0
```

Университет Сириус Колледж

Области видимости. Оператор `global`

```
1  animal = 'cat'
2
3  def print_global():
4      print('global variable animal: ', animal)
5
6  print_global()
7  # global variable animal: cat
```

Университет Сириус Колледж

Области видимости. Оператор `global`

```
1  animal = 'cat'
2
3  def change_and_print_global():
4      print('global variable animal: ', animal)
5
6      animal = 'dog'
7
8      print('after change: ', animal)
9
10 change_and_print_global()
11 # UnboundLocalError: cannot access local variable 'animal' where it is
12 # associated with a value
```

Университет Сириус Колледж

Области видимости. Оператор `global`

```
1  animal = 'cat'
2
3  def change_and_print_global():
4      animal = 'dog'
5
6      print('after change: ', animal)
7
8  change_and_print_global()
9  print('global variable animal: ', animal)
10 # after change: dog
11 # global variable animal: cat
```

Университет Сириус Колледж

Области видимости. Оператор `global`

```
1  animal = 'cat'
2
3  def change_and_print_global():
4      global animal
5      animal = 'dog'
6
7      print('after change: ', animal)
8
9  change_and_print_global()
10 print('global variable animal: ', animal)
11 # after change: dog
12 # global variable animal: dog
```

Университет Сириус Колледж

Замыкания

Замыкание (closure) — это функция, которая запоминает значения из своей внешней области видимости, даже если эта область уже недоступна. Она создается, когда функция объявляется, и продолжает запоминать значения переменных даже после того, как вызывающая функция завершит свою работу.

Замыкания — это инструмент, который позволяет сохранять значения и состояние между вызовами функций, создавать функции на лету и возвращать их из других функций.

Университет Сириус Колледж

Замыкания

```python
def outer_function(x):
    def inner_function(y):
        return x + y
    return inner_function

closure = outer_function(10)
print(closure(5))
# 15
```

Университет Сириус Колледж

Замыкания

```python
def counter():
    count = 0
    def inner():
        nonlocal count
        count += 1
        return count
    return inner

c = counter()
print(c())  # 1
print(c())  # 2
print(c())  # 3
```

Университет Сириус Колледж

Замыкания

```
1  def add_number(n):
2      def inner(x):
3          return x + n
4      return inner
5
6  add_five = add_number(5)
7  add_ten = add_number(10)
8
9  print(add_five(3))  # 8
10  print(add_ten(3))   # 13
```

Университет Сириус Колледж

Замыкания

```
def password_protected(password):
    def inner():
        if password == 'secret':
            print("Access granted")
        else:
            print("Access denied")
    return inner

login = password_protected('secret')
 login()  # Access granted
```

Замыкания

```python
config = {
    'language': 'ru',
    'timezone': 'UTC'
}

def get_config(key):
    def inner():
        return config.get(key, None)
    return inner

get_language = get_config('language')
get_timezone = get_config('timezone')

print(get_language())  # ru
print(get_timezone())  # UTC
```

Университет Сириус Колледж

Генераторы

Функция считается генератором, если:

- Содержит одно или несколько выражений `yield`.
- При вызове возвращает объект типа generator, но не начнет выполнение.
- Методы `__iter__()` и `__next__()` реализуются автоматически.
- После каждого вызова функция приостанавливается, а управление передается вызывающей стороне.
- Локальные переменные и их состояния запоминаются между последовательными вызовами.
- Когда вычисления заканчиваются по какому то условию, автоматически вызывается `StopIteration`.

Университет Сириус Колледж

Генераторы

```
1  def range_1(first=0, last=5, step=1):
2      number = first
3      while number < last:
4          yield number
5          number += step
```

Университет Сириус Колледж

Генераторы

```
1  def range_1(first=0, last=5, step=1):
2      number = first
3      while number < last:
4          yield number
5          number += step
```

Университет Сириус Колледж

Генераторы

```
1  def range_1(first=0, last=5, step=1):
2      number = first
3      while number < last:
4          yield number
5          number += step
```

Университет Сириус Колледж

Генераторы

```
1  def range_1(first=0, last=5, step=1):
2      number = first
3      while number < last:
4          yield number
5          number += step
6
7
8  print(type(range_1))
9  # <class 'function'>
```

Университет Сириус Колледж

Генераторы

```python
def range_1(first=0, last=5, step=1):
    number = first
    while number < last:
        yield number
        number += step


print(type(range_1))
# <class 'function'>
ranger = range_1()
print(type(ranger))
# <class 'generator'>
```

Университет Сириус Колледж

Генераторы

```python
def range_1(first=0, last=5, step=1):
    number = first
    while number < last:
        yield number
        number += step


print(type(range_1))
# <class 'function'>
ranger = range_1()
print(type(ranger))
# <class 'generator'>
for i in ranger:
    print(i)
```

Университет Сириус Колледж

Генераторы

```
1  def range_1(first=0, last=2, step=1):
2      number = first
3      while number < last:
4          yield number
5          number += step
```

Университет Сириус Колледж

Генераторы

```
1  def range_1(first=0, last=2, step=1):
2      number = first
3      while number < last:
4          yield number
5          number += step
6
7  ranger = range_1()
8  print(next(ranger))
9  # 0
```

Университет Сириус Колледж

Генераторы

```
def range_1(first=0, last=2, step=1):
    number = first
    while number < last:
        yield number
        number += step

ranger = range_1()
print(next(ranger))
print(next(ranger))
 # 1
```

Университет Сириус Колледж

Генераторы

```
1  def range_1(first=0, last=2, step=1):
2      number = first
3      while number < last:
4          yield number
5          number += step
6
7  ranger = range_1()
8  print(next(ranger))
9  print(next(ranger))
10 print(next(ranger))
11 # Traceback (most recent call last):
12 #   File "<stdin>", line 10, in <module>
13 #     print(next(ranger))
14 # StopIteration
```

Университет Сириус Колледж

Генераторы. Примеры.

```
1  # Чтение большого файла
2  def csv_reader(file_name):
3      file = open(file_name)
4      result = file.read().split("\n")
5      return result
```

Университет Сириус Колледж

Генераторы. Примеры.

```python
# Чтение большого файла
def csv_reader(file_name):
    for row in open(file_name, "r"):
        yield row
```

Университет Сириус Колледж

Генераторы. Примеры.

```python
# Бесконечный генератор
def infinite_sequence():
    num = 0
    while True:
        yield num
        num += 1
```

Университет Сириус Колледж

Генераторные выражения (generator expressions, generator comprehensions)

```
1  nums_squared_lc = [num**2 for num in range(5)]
2  print(type(nums_squared_lc))
3  # <class 'list'>
```

Университет Сириус Колледж

Генераторные выражения (generator expressions, generator comprehensions)

```python
nums_squared_lc = [num**2 for num in range(5)]
print(type(nums_squared_lc))
# <class 'list'>
nums_squared_gc = (num**2 for num in range(5))
print(type(nums_squared_gc))
# <class 'generator'>
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
1  import sys
2
3
4  nums_squared_lc = [i ** 2 for i in range(10000)]
```

Университет Сириус Колледж

Сравнение генераторов и списков

```python
import sys


nums_squared_lc = [i ** 2 for i in range(10000)]
sys.getsizeof(nums_squared_lc)
# 87624
```

Университет Сириус Колледж

Сравнение генераторов и списков

```python
import sys


nums_squared_lc = [i ** 2 for i in range(10000)]
sys.getsizeof(nums_squared_lc)
# 87624
nums_squared_gc = (i ** 2 for i in range(10000))
print(sys.getsizeof(nums_squared_gc))
# 120
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
1  import cProfile
2
3  cProfile.run('sum([i * 2 for i in range(10000)])')
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
import cProfile

cProfile.run('sum([i * 2 for i in range(10000)])')
#          5 function calls in 0.001 seconds
#
#    Ordered by: standard name
#
#    ncalls  tottime  percall  cumtime  percall filename:lineno(functior
#         1    0.001    0.001    0.001    0.001 <string>:1(<listcomp>)
#         1    0.000    0.000    0.001    0.001 <string>:1(<module>)
#         1    0.000    0.000    0.001    0.001 {built-in method builtir
#         1    0.000    0.000    0.000    0.000 {built-in method builtir
#         1    0.000    0.000    0.000    0.000 {method 'disable' of '_l
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
1  import cProfile
2
3  cProfile.run('sum([i * 2 for i in range(10000)])')
4  #          5 function calls in 0.001 seconds
5  #
6  #    Ordered by: standard name
7  #
8  #    ncalls  tottime  percall  cumtime  percall filename:lineno(functior
9  #         1    0.001    0.001    0.001    0.001 <string>:1(<listcomp>)
10 #         1    0.000    0.000    0.001    0.001 <string>:1(<module>)
11 #         1    0.000    0.000    0.001    0.001 {built-in method builtir
12 #         1    0.000    0.000    0.000    0.000 {built-in method builtir
13 #         1    0.000    0.000    0.000    0.000 {method 'disable' of '_l
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
import cProfile

cProfile.run('sum([i * 2 for i in range(10000)])')
#          5 function calls in 0.001 seconds
#
#    Ordered by: standard name
#
#    ncalls  tottime  percall  cumtime  percall filename:lineno(functior
#         1    0.001    0.001    0.001    0.001 <string>:1(<listcomp>)
#         1    0.000    0.000    0.001    0.001 <string>:1(<module>)
#         1    0.000    0.000    0.001    0.001 {built-in method builtir
#         1    0.000    0.000    0.000    0.000 {built-in method builtir
#         1    0.000    0.000    0.000    0.000 {method 'disable' of '_l
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
1  import cProfile
2
3  cProfile.run('sum([i * 2 for i in range(10000)])')
4  #          5 function calls in 0.001 seconds
5  # ...
6
7  cProfile.run('sum((i * 2 for i in range(10000)))')
8  #          10005 function calls in 0.003 seconds
9  # ...
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
1  import cProfile
2
3  cProfile.run('sum([i * 2 for i in range(10000)])')
4  #          5 function calls in 0.001 seconds
5  #  ...
6
7  cProfile.run('sum((i * 2 for i in range(10000)))')
8  #          10005 function calls in 0.003 seconds
9  #  ...
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
1  import cProfile
2
3  cProfile.run('sum([i * 2 for i in range(10000)])')
4  #          5 function calls in 0.001 seconds
5  # ...
6
7  cProfile.run('sum((i * 2 for i in range(10000)))')
8  #          10005 function calls in 0.003 seconds
9  # ...
```

Университет Сириус Колледж

Сравнение генераторов и списков

```
1  import cProfile
2
3  cProfile.run('sum([i * 2 for i in range(10000)])')
4  #          5 function calls in 0.001 seconds
5  # ...
6
7  cProfile.run('sum((i * 2 for i in range(10000)))')
8  #          10005 function calls in 0.003 seconds
9  # ...
```

Генераторы хоть и дают существенное преимущество в объеме памяти, могут работать значительно медленнее, чем списки.

Университет Сириус Колледж

Генераторы. Методы .send(), .throw(), .close()

```python
def is_palindrome(num):
    if num // 10 == 0:
        return False
    temp = num
    reversed_num = 0

    while temp != 0:
        reversed_num = (reversed_num * 10) + (temp % 10)
        temp = temp // 10

    if num == reversed_num:
        return True
    else:
        return False
```

Университет Сириус Колледж

Генераторы. Методы `.send()`, `.throw()`, `.close()`

```
def infinite_palindromes():
    num = 0
    while True:
        if is_palindrome(num):
            i = (yield num)
            if i is not None:
                num = i
        num += 1
```

Университет Сириус Колледж

Генераторы. Методы `.send()`, `.throw()`, `.close()`

```
1  def infinite_palindromes():
2      num = 0
3      while True:
4          if is_palindrome(num):
5              i = (yield num)
6              if i is not None:
7                  num = i
8          num += 1
```

`i` принимает значение возвращаемое `yield`. Это позволяет изменять полученное значение. И, что важнее, позволяет отправлять (`.send()`) новое значение в генератор. Когда выполнение продолжается после `yield`, `i` получит отправленное значение.

Университет Сириус Колледж

Генераторы. Методы `.send()`, `.throw()`, `.close()`

```
1  def infinite_palindromes():
2      num = 0
3      while True:
4          if is_palindrome(num):
5              i = (yield num)
6              if i is not None:
7                  num = i
8          num += 1
```

`i` принимает значение возвращаемое `yield`. Это позволяет изменять полученное значение. И, что важнее, позволяет отправлять (`.send()`) новое значение в генератор. Когда выполнение продолжается после `yield`, `i` получит отправленное значение.

Университет Сириус Колледж

Генераторы. Методы .send(), .throw(), .close()

```
1  def infinite_palindromes():
2      num = 0
3      while True:
4          if is_palindrome(num):
5              i = (yield num)
6              if i is not None:
7                  num = i
8          num += 1
9
10 pal_gen = infinite_palindromes()
11 for i in pal_gen:
12     digits = len(str(i))
13     pal_gen.send(10 ** (digits))
```

Университет Сириус Колледж

Генераторы. Методы `.send()`, `.throw()`, `.close()`

```
1  def infinite_palindromes():
2      num = 0
3      while True:
4          if is_palindrome(num):
5              i = (yield num)
6              if i is not None:
7                  num = i
8          num += 1
```

Генератор `infinite_palindromes` является корутиной.

Университет Сириус Колледж

Генераторы. Методы .send(), .throw(), .close()

```
pal_gen = infinite_palindromes()
for i in pal_gen:
    print(i)
    digits = len(str(i))
    if digits == 5:
        pal_gen.throw(ValueError("We don't like large palindromes"))
    pal_gen.send(10 ** (digits))
```

Университет Сириус Колледж

Генераторы. Методы `.send()`, `.throw()`, `.close()`

```
1  pal_gen = infinite_palindromes()
2  for i in pal_gen:
3      print(i)
4      digits = len(str(i))
5      if digits == 5:
6          pal_gen.throw(ValueError("We don't like large palindromes"))
7      pal_gen.send(10 ** (digits))
```

`.throw()` Создает исключение в точке, где генератор был приостановлен, и возвращает следующее значение, выданное функцией генератора . Если генератор завершает работу, не выдав другого значения, то возникает исключение StopIteration. Если функция генератора не улавливает переданное исключение или создает другое исключение, то это исключение распространяется на вызывающую сторону/программу.

Университет Сириус Колледж

Генераторы. Методы `.send()`, `.throw()`, `.close()`

```
1  pal_gen = infinite_palindromes()
2  for i in pal_gen:
3      print(i)
4      digits = len(str(i))
5      if digits == 5:
6          pal_gen.close()
7      pal_gen.send(10 ** (digits))
```

Университет Сириус Колледж

Генераторы. Методы `.send()`, `.throw()`, `.close()`

```
1  pal_gen = infinite_palindromes()
2  for i in pal_gen:
3      print(i)
4      digits = len(str(i))
5      if digits == 5:
6          pal_gen.close()
7      pal_gen.send(10 ** (digits))
```

Метод `.close()` прерывает выполнение генератора.

Университет Сириус Колледж

Декораторы

Вспомним, как работают функции:

Университет Сириус Колледж

Декораторы

Вспомним, как работают функции:

- Функции являются объектами первого класса. Это означает, что функции можно передавать и использовать в качестве аргументов.

Университет Сириус Колледж

Декораторы

Вспомним, как работают функции:

- Функции являются объектами первого класса. Это означает, что функции можно передавать и использовать в качестве аргументов.
- Можно определить функции внутри других функций.

Университет Сириус Колледж

Декораторы

Вспомним, как работают функции:

- Функции являются объектами первого класса. Это означает, что функции можно передавать и использовать в качестве аргументов.
- Можно определить функции внутри других функций.
- Функции умеют возвращать другие функции в качестве результата.

Декораторы

```python
def sample_decorator(func):
    def wrapper():
        print('Я родился...')
        func()
        print('Меня зовут Лунтик!')
    return wrapper
```

Декораторы

```python
def sample_decorator(func):
    def wrapper():
        print('Я родился...')
        func()
        print('Меня зовут Лунтик!')
    return wrapper
```

Университет Сириус Колледж

Декораторы

```
1  def sample_decorator(func):
2      def wrapper():
3          print('Я родился...')
4          func()
5          print('Меня зовут Лунтик!')
6      return wrapper
```

Университет Сириус Колледж

Декораторы

```
1  def sample_decorator(func):
2      def wrapper():
3          print('Я родился...')
4          func()
5          print('Меня зовут Лунтик!')
6      return wrapper
```

Декораторы

```python
def sample_decorator(func):
    def wrapper():
        print('Я родился...')
        func()
        print('Меня зовут Лунтик!')
    return wrapper
```

Декораторы

```python
def sample_decorator(func):
    def wrapper():
        print('Я родился...')
        func()
        print('Меня зовут Лунтик!')
    return wrapper

def say():
    print('Привет Мир.')

say = sample_decorator(say)
```

Университет Сириус Колледж

Декораторы

```
1  def sample_decorator(func):
2      def wrapper():
3          print('Я родился...')
4          func()
5          print('Меня зовут Лунтик!')
6      return wrapper
7
8  def say():
9      print('Привет Мир.')
10
11 say = sample_decorator(say)
```

Декораторы

```python
def sample_decorator(func):
    def wrapper():
        print('Я родился...')
        func()
        print('Меня зовут Лунтик!')
    return wrapper

def say():
    print('Привет Мир.')

say = sample_decorator(say)
say
# <function sample_decorator.<locals>.wrapper at 0x7f591a0a42f0>
```

Университет Сириус Колледж

Декораторы

```
1  def sample_decorator(func):
2      def wrapper():
3          print('Я родился...')
4          func()
5          print('Меня зовут Лунтик!')
6      return wrapper
7
8  def say():
9      print('Привет Мир.')
10
11 say = sample_decorator(say)
12 say
13 # <function sample_decorator.<locals>.wrapper at 0x7f591a0a42f0>
```

Декораторы

```python
def sample_decorator(func):
    def wrapper():
        print('Я родился...')
        func()
        print('Меня зовут Лунтик!')
    return wrapper

def say():
    print('Привет Мир.')

say = sample_decorator(say)
say()
# Я родился...
# Привет Мир.
# Меня зовут Лунтик!
```

Университет Сириус Колледж

Декораторы

```
1  def sample_decorator(func):
2      def wrapper():
3          print('Я родился...')
4          func()
5          print('Меня зовут Лунтик!')
6      return wrapper
7
8  @sample_decorator
9  def say():
10     print('Привет Мир.')
11
12 say
13 # <function sample_decorator.<locals>.wrapper at 0x789564722f20>
```

Университет Сириус
Колледж

Проще говоря: декораторы обертывают функцию, изменяя ее поведение.

Университет Сириус Колледж

Декорирование функций с аргументами

```
1  def do_twice(func):
2      def wrapper_do_twice():
3          func()
4          func()
5      return wrapper_do_twice
```

Университет Сириус Колледж

Декорирование функций с аргументами

```
1  def do_twice(func):
2      def wrapper_do_twice():
3          func()
4          func()
5      return wrapper_do_twice
6
7  @do_twice
8  def say_whee():
9      print("Whee!")
```

Университет Сириус Колледж

Декорирование функций с аргументами

```
1  def do_twice(func):
2      def wrapper_do_twice():
3          func()
4          func()
5      return wrapper_do_twice
6
7  @do_twice
8  def say_whee():
9      print("Whee!")
10
11 say_whee()
12 # Whee!
13 # Whee!
```

Университет Сириус Колледж

Декорирование функций с аргументами

```python
def do_twice(func):
    def wrapper_do_twice():
        func()
        func()
    return wrapper_do_twice

@do_twice
def greet(name):
    print(f"Hello {name}")
```

Университет Сириус Колледж

Декорирование функций с аргументами

```python
def do_twice(func):
    def wrapper_do_twice():
        func()
        func()
    return wrapper_do_twice

@do_twice
def greet(name):
    print(f"Hello {name}")

greet(name="World")
# Traceback (most recent call last):
#   ...
# TypeError: wrapper_do_twice() takes 0 positional arguments but 1 was
```

Университет Сириус Колледж

Декорирование функций с аргументами

```python
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        func(*args, **kwargs)
    return wrapper_do_twice

@do_twice
def greet(name):
    print(f"Hello {name}")

greet("World")
# Hello World
# Hello World
```

Университет Сириус Колледж

Декорирование функций с аргументами

```python
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        func(*args, **kwargs)
    return wrapper_do_twice

@do_twice
def greet(name):
    print(f"Hello {name}")

greet("World")
# Hello World
# Hello World
```

Университет Сириус Колледж

Декорирование функций с аргументами

```python
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        func(*args, **kwargs)
    return wrapper_do_twice

@do_twice
def greet(name):
    print(f"Hello {name}")

greet("World")
# Hello World
# Hello World
```

Университет Сириус Колледж

Возвращаемые значения из декорированных функций

```
@do_twice
def return_greeting(name):
    print("Creating greeting")
    return f"Hi {name}"
```

Университет Сириус Колледж

Возвращаемые значения из декорированных функций

```python
@do_twice
def return_greeting(name):
    print("Creating greeting")
    return f"Hi {name}"

hi_adam = return_greeting("Adam")
# Creating greeting
# Creating greeting

print(hi_adam)
# None
```

Университет Сириус Колледж

Возвращаемые значения из декорированных функций

```python
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

Университет Сириус Колледж

Возвращаемые значения из декорированных функций

```python
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

Университет Сириус Колледж

Возвращаемые значения из декорированных функций

```python
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

Университет Сириус Колледж

Возвращаемые значения из декорированных функций

```python
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice

@do_twice
def return_greeting(name):
    print("Creating greeting")
    return f"Hi {name}"

return_greeting("Adam")
# Creating greeting
# Creating greeting
# 'Hi Adam'
```

Университет Сириус Колледж

@functools.wraps

```
print
# <built-in function print>
print.__name__
# 'print'
help(print)
# Help on built-in function print in module builtins:
#
# print(...)
#     <full help message>
```

Университет Сириус Колледж

@functools.wraps

```
say_whee
# <function do_twice.<locals>.wrapper_do_twice at 0x7f43700e52f0>
say_whee.__name__
# 'wrapper_do_twice'
help(say_whee)
# Help on function wrapper_do_twice in module decorators:
#
# wrapper_do_twice()
```

Университет Сириус Колледж

@functools.wraps

```python
import functools

def do_twice(func):
    @functools.wraps(func)
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

Университет Сириус Колледж

@functools.wraps

```python
import functools

def do_twice(func):
    @functools.wraps(func)
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

Университет Сириус Колледж

@functools.wraps

```python
import functools

def do_twice(func):
    @functools.wraps(func)
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        return func(*args, **kwargs)
    return wrapper_do_twice
```

@functools.wraps

```
@do_twice
def say_whee():
    print("Whee!")


say_whee
# <function say_whee at 0x7ff79a60f2f0>
say_whee.__name__
# 'say_whee'
help(say_whee)
# Help on function say_whee in module whee:
#
# say_whee()
```

Университет Сириус Колледж

Декораторы. Примеры

Базовый шаблон декоратора:

```
import functools

def decorator(func):
    @functools.wraps(func)
    def wrapper_decorator(*args, **kwargs):
        # Do something before
        value = func(*args, **kwargs)
        # Do something after
        return value
     return wrapper_decorator
```

Университет Сириус Колледж

Декораторы. Примеры

Измерение времени выполнения:

```python
import functools
import time

def timer(func):
    """Print the runtime of the decorated function"""
    @functools.wraps(func)
    def wrapper_timer(*args, **kwargs):
        start_time = time.perf_counter()
        value = func(*args, **kwargs)
        end_time = time.perf_counter()
        run_time = end_time - start_time
        print(f"Finished {func.__name__}() in {run_time:.4f} secs")
        return value
    return wrapper_timer
```

Университет Сириус Колледж

Декораторы. Примеры

Измерение времени выполнения:

```python
import functools
import time

def timer(func): ...

@timer
def waste_some_time(num_times):
    for _ in range(num_times):
        sum([number**2 for number in range(10_000)])

waste_some_time(1)
# Finished waste_some_time() in 0.0010 secs
waste_some_time(999)
# Finished waste_some_time() in 0.3260 secs
```

Университет Сириус Колледж

Декораторы. Примеры

Отладка:

```python
def debug(func):
    """Print the function signature and return value"""
    @functools.wraps(func)
    def wrapper_debug(*args, **kwargs):
        args_repr = [repr(a) for a in args]
        kwargs_repr = [f"{k}={repr(v)}" for k, v in kwargs.items()]
        signature = ", ".join(args_repr + kwargs_repr)
        print(f"Calling {func.__name__}({signature})")
        value = func(*args, **kwargs)
        print(f"{func.__name__}() returned {repr(value)}")
        return value
    return wrapper_debug
```

Университет Сириус Колледж

Декораторы. Примеры

Отладка:

```
@debug
def make_greeting(name, age=None):
    if age is None:
        return f"Howdy {name}!"
    else:
        return f"Whoa {name}! {age} already, you're growing up!"

make_greeting("Benjamin")
# Calling make_greeting('Benjamin')
# make_greeting() returned 'Howdy Benjamin!'
# 'Howdy Benjamin!'
```

Университет Сириус Колледж

Декораторы. Примеры

Отладка:

```
@debug
def make_greeting(name, age=None):
    if age is None:
        return f"Howdy {name}!"
    else:
        return f"Whoa {name}! {age} already, you're growing up!"

make_greeting("Juan", age=114)
# Calling make_greeting('Juan', age=114)
# make_greeting() returned 'Whoa Juan! 114 already, you're growing up!'
# 'Whoa Juan! 114 already, you're growing up!'
```

Университет Сириус Колледж

Декораторы. Примеры

Отладка:

```python
@debug
def make_greeting(name, age=None):
    if age is None:
        return f"Howdy {name}!"
    else:
        return f"Whoa {name}! {age} already, you're growing up!"

make_greeting(name="Maria", age=116)
# Calling make_greeting(name='Maria', age=116)
# make_greeting() returned 'Whoa Maria! 116 already, you're growing up!
# 'Whoa Maria! 116 already, you're growing up!'
```

Университет Сириус Колледж

Декораторы. Примеры

Замедление времени выполнения:

```
import functools
import time

def slow_down(func):
    """Sleep 1 second before calling the function"""
    @functools.wraps(func)
    def wrapper_slow_down(*args, **kwargs):
        time.sleep(1)
        return func(*args, **kwargs)
     return wrapper_slow_down
```

Университет Сириус Колледж

Декораторы. Примеры

Регистрация функций:

```
PLUGINS = dict()

def register(func):
    """Register a function as a plug-in"""
    PLUGINS[func.__name__] = func
    return func
```

Университет Сириус Колледж

Декораторы. Примеры

```
1  PLUGINS = dict()
2
3  def register(func): ...
4
5  @register
6  def say_hello(name):
7      return f"Hello {name}"
8
9  @register
10 def be_awesome(name):
11     return f"Yo {name}, together we're the awesomest!"
12
13 PLUGINS
14 # {'say_hello': <function say_hello at 0x7f768eae6730>,
15 #  'be_awesome': <function be_awesome at 0x7f768eae67b8>}
```

Университет Сириус Колледж

Прочие возможности декораторов

```
@debug
@do_twice
def greet(name):
    print(f"Hello {name}")
```

Университет Сириус Колледж

Прочие возможности декораторов

```
@repeat(num_times=4)
def greet(name):
    print(f"Hello {name}")


greet("World")
# Hello World
# Hello World
# Hello World
 # Hello World
```

Университет Сириус Колледж

Прочие возможности декораторов

```
def repeat(num_times):
    def decorator_repeat(func):
        @functools.wraps(func)
        def wrapper_repeat(*args, **kwargs):
            for _ in range(num_times):
                value = func(*args, **kwargs)
            return value
        return wrapper_repeat
    return decorator_repeat
```

Университет Сириус Колледж

Прочие возможности декораторов

```python
def repeat(num_times):
    def decorator_repeat(func):
        @functools.wraps(func)
        def wrapper_repeat(*args, **kwargs):
            for _ in range(num_times):
                value = func(*args, **kwargs)
            return value
        return wrapper_repeat
    return decorator_repeat
```

Университет Сириус Колледж

Прочие возможности декораторов

```python
def repeat(num_times):
    def decorator_repeat(func):
        @functools.wraps(func)
        def wrapper_repeat(*args, **kwargs):
            for _ in range(num_times):
                value = func(*args, **kwargs)
            return value
        return wrapper_repeat
    return decorator_repeat
```

Университет Сириус Колледж

Прочие возможности декораторов

```
def repeat(num_times):
    def decorator_repeat(func):
        @functools.wraps(func)
        def wrapper_repeat(*args, **kwargs):
            for _ in range(num_times):
                value = func(*args, **kwargs)
            return value
        return wrapper_repeat
    return decorator_repeat
```

Университет Сириус Колледж

Прочие возможности декораторов

```
def repeat(num_times):
    def decorator_repeat(func):
        @functools.wraps(func)
        def wrapper_repeat(*args, **kwargs):
            for _ in range(num_times):
                value = func(*args, **kwargs)
            return value
        return wrapper_repeat
    return decorator_repeat
```

Университет Сириус Колледж

Прочие возможности декораторов

```
def name(_func=None, *, key1=value1, key2=value2, ...):
    def decorator_name(func):
        ...  # Create and return a wrapper function.

    if _func is None:
        return decorator_name
    else:
        return decorator_name(_func)
```

Университет Сириус Колледж

Прочие возможности декораторов

```python
def repeat(_func=None, *, num_times=2):
    def decorator_repeat(func):
        @functools.wraps(func)
        def wrapper_repeat(*args, **kwargs):
            for _ in range(num_times):
                value = func(*args, **kwargs)
            return value
        return wrapper_repeat

    if _func is None:
        return decorator_repeat
    else:
        return decorator_repeat(_func)
```

Университет Сириус Колледж

Прочие возможности декораторов

```
def repeat(_func=None, *, num_times=2): ...

@repeat
def say_whee():
    print("Whee!")

@repeat(num_times=3)
def greet(name):
    print(f"Hello {name}")
```

Университет Сириус Колледж

Прочие возможности декораторов

```
say_whee()
# Whee!
# Whee!

greet("Penny")
# Hello Penny
# Hello Penny
# Hello Penny
```

Университет Сириус Колледж

Прочие возможности декораторов

```python
def count_calls(func):
    @functools.wraps(func)
    def wrapper_count_calls(*args, **kwargs):
        wrapper_count_calls.num_calls += 1
        print(f"Call {wrapper_count_calls.num_calls} of {func.__name__}(
        return func(*args, **kwargs)
    wrapper_count_calls.num_calls = 0
    return wrapper_count_calls
```

Университет Сириус Колледж

Прочие возможности декораторов

```
@count_calls
def say_whee():
    print("Whee!")

say_whee()
# Call 1 of say_whee()
# Whee!

say_whee()
# Call 2 of say_whee()
# Whee!

say_whee.num_calls
# 2
```

Университет Сириус Колледж

Обработка ошибок

- Ошибки отображаются с помощью специальных возвращаемых значений. В Python для отслеживания и корректной обработки ошибок используются исключения.
- **Исключение** - это такой код, который выполняется, когда происходит связанная с ним ошибка.
- Когда вы выполняете код, в котором при некоторых обстоятельствах могут возникнуть ошибки, вам понадобятся **обработчики исключений**.
- Если не предоставить Python код обработчика ошибок, выведется сообщение об ошибке, а программа завершится.

Обработка ошибок

```
1  short_list = [1, 2, 3]
2
3  position = 3
4
5  short_list[position]
6  # IndexError: list index out of range
```

Университет Сириус Колледж

Обработка ошибок

```python
short_list = [1, 2, 3]

position = 3

try:
    short_list[position]
except:
    print(f"Need position between 0 and {len(short_list) - 1}")
# Need position between 0 and 2
```

Университет Сириус Колледж

Обработка ошибок

```
short_list = [1, 2, 3]

while True:
    value = input("Position? [q to quit] ")
    if value == 'q':
        break
    try:
        position = int(value)
        print(short_list[position])
    except IndexError as err:
        print(f"Bad index: {err}")
    except Exception as other:
        print(f"Something else broke: {other}")
```

Университет Сириус Колледж

Обработка ошибок

```
print("You can get 3 squares if you'd like")
for i in range(3):
    try:
        number = int(input())
    except ValueError:
        print("That's not a number")
    except KeyboardInterrupt:
        print("How rude of you")
    else:
         print(f"{number} squared is {number**2}")
```

Университет Сириус Колледж

Обработка ошибок

```python
print("You can get 3 squares if you'd like")
for i in range(3):
    try:
        number = int(input())
    except ValueError:
        print("That's not a number")
    except KeyboardInterrupt:
        print("How rude of you")
    else:
        print(f"{number} squared is {number**2}")
    finally:
        print("This always runs")
```