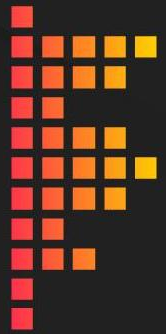


Что будет на экране?

```
x = 1
y = x / 3
z = y * 3

if x == z:
    print("Trump")
else:
    print("Biden")
```

ML Basic

Функции.Часть 2 (декораторы)

otus.ru

REC Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо “-”,
если есть проблемы

Тема вебинара

ML Basic
Функции.Часть 2 (декораторы)

Константин Алексин

Специалист по прогнозному моделированию

Улучшаю банковские продукты и процессы с использованием AI-моделей и подходов Causal Inference.
Преподаю дисциплину “Интерпретируемые модели” в РАНХИГС.

Финалист Хакатона «Лидеры цифровой трансформации 2020».
Спикер на конференции AI Journey 2019 «Обработка аудио с помощью Deep Learning».

Если запись не идет,
скажите мне

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо “-”,
если есть проблемы

Правила вебинара

Активно участвуем

Off-topic обсуждаем в учебной группе

Задаем вопрос в чат

Вопросы вижу в чате, могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или задайте вопрос

Что будет на экране?

```
x = 1
y = x / 3
z = y * 3

if x == z:
        print("Trump")
else:
        print("Biden")
```

Что будет на экране?

```
x = 1
y = x / 3 # 0.333333333
z = y * 3

if x == z:
        print("Trump")
else:
        print("Biden")
```

Что будет на экране?

```
x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if x == z:
        print("Trump")
else:
        print("Biden")
```

Посложнее

```
from math import sin, asin, sqrt

x = 1
y = sin(sin(x) ** 2)
z = asin(sqrt(asin(y)))  # 1.0000000000000002

if x == z:
 print("Trump")
else:
 print("Biden")
```

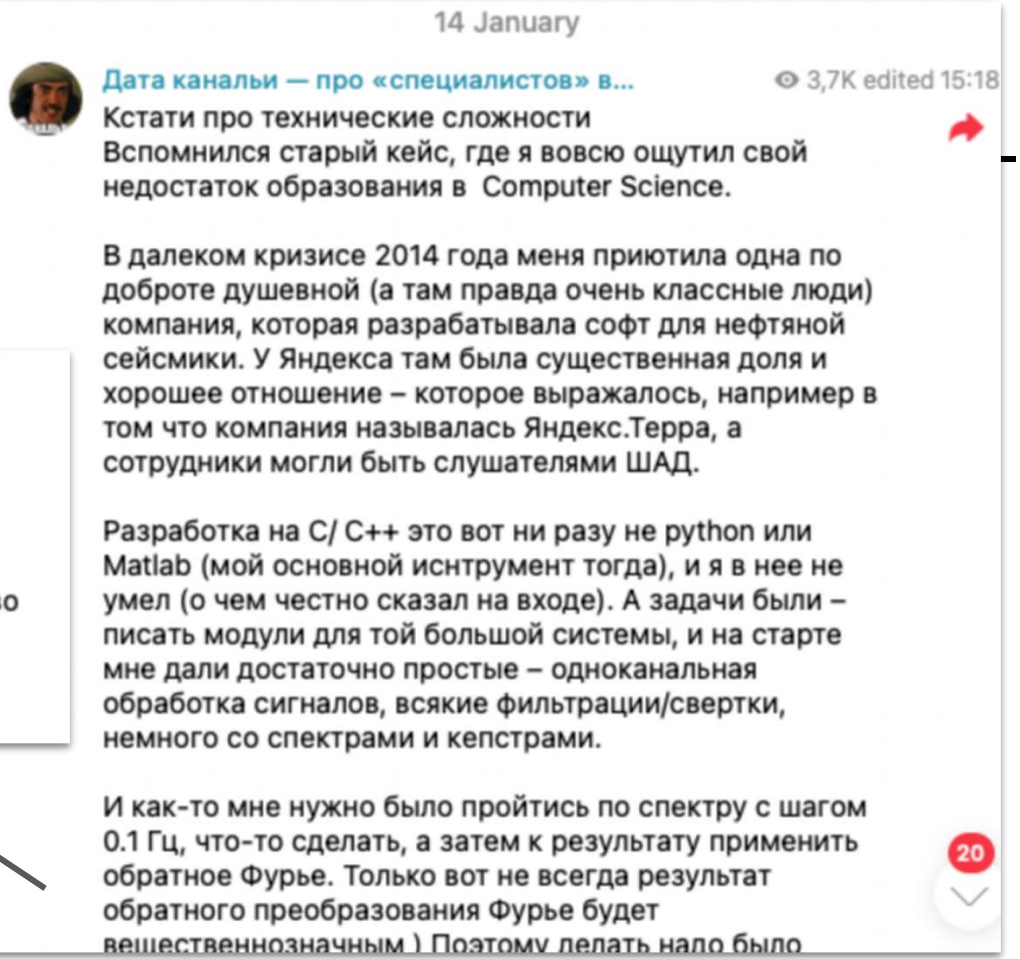
Что будет на экране?

```
x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if x == z:
        print("Trump")

EPS = 0.000000001

if abs(x - z) < EPS:
        print("Trump")
```

История

https://t.me/datarascals/116

Несколько дней я потратил, пытаясь добиться того же результата в C – без шанса 🙈🤯.
В матлабе же не только индексация массивов отличается)
В итоге пошел на поклон к синьору и тут вскрылся мой недостаток образования на тот момент в CS. Что-то о свойствах вещественных чисел я знал (что на равенство сравнивать нельзя, ибо хранятся они в некотором приближении), но вот глубоко не копал – на чем и погорел.



Вынесем в функцию

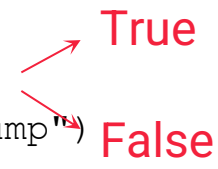
```
x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if x == z:
        print("Trump")

EPS = 0.000000001

if abs(x - z) < EPS:
        print("Trump")
else:
        print("Biden")
```

Вынесем в функцию

```
def is_equal(a: float, b: float) -> bool:
  …

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
```

Вынесем в функцию

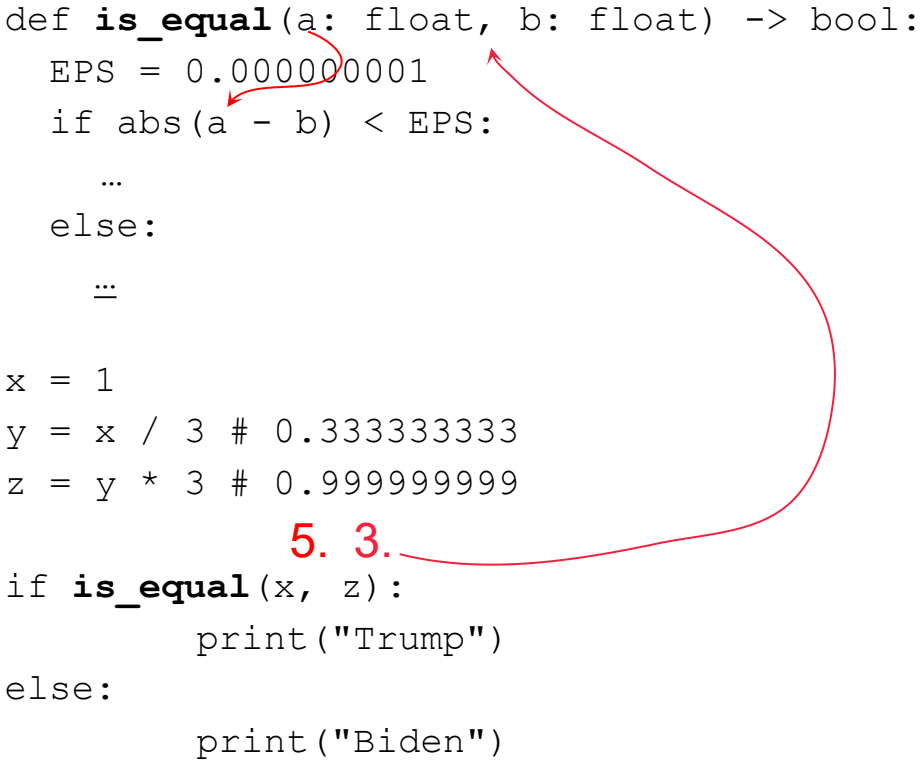
```
def is_equal(a: float, b: float) -> bool:
  …

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
```



Вынесем в функцию

```
def is_equal(a: float, b: float) -> bool:
  EPS = 0.000000001
  if abs(a - b) < EPS:
    …
  else:
    …

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
```

Вынесем в функцию



Вынесем в функцию

```
def is_equal(a: float, b: float) -> bool:
  EPS = 0.000000001
  if abs(a - b) < EPS:
        return True
  else:
        return False

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
```

Вынесем в функцию

```
def is_equal(a: float, b: float) -> bool:
  EPS = 0.000000001
  return abs(a - b) < EPS

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
```

Вынесем константу

```
def is_equal(a: float, b: float) -> bool:
    EPS = 0.000000001
    return abs(a - b) < EPS

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
```

Что возможно?

```
def is_equal(a: float, b: float) ->
bool:
  EPS = 0.000000001
  return abs(a - b) < EPS

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
print("Мы сравнивали с точностью до ", EPS)
```

1

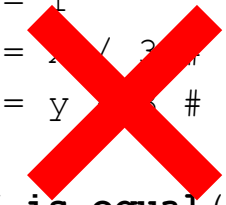
```
def is_equal(a: float, b: float) ->
bool:
    return abs(a - b) < EPS

x = 1
y = x / 3 # 0.333333333
z = y * 3 # 0.999999999

EPS = 0.000000001
if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
print("Мы сравнивали с точностью до ", EPS)
```

2

Области видимости

- Область видимости:
  - Локальная
  - Глобальная
- Локальная переменная
- Глобальная переменная

```
def is_equal(a: float, b: float) ->
bool:
  EPS = 0.000000001
  return abs(a - b) < EPS

x = 1
y = 1 / 3  # 0.333333333
z = y + y + y  # 0.999999999

if is_equal(x, z):
        print("Trump")
else:
        print("Biden")
print("Мы сравнивали с точностью до ", EPS)
```



THEORY

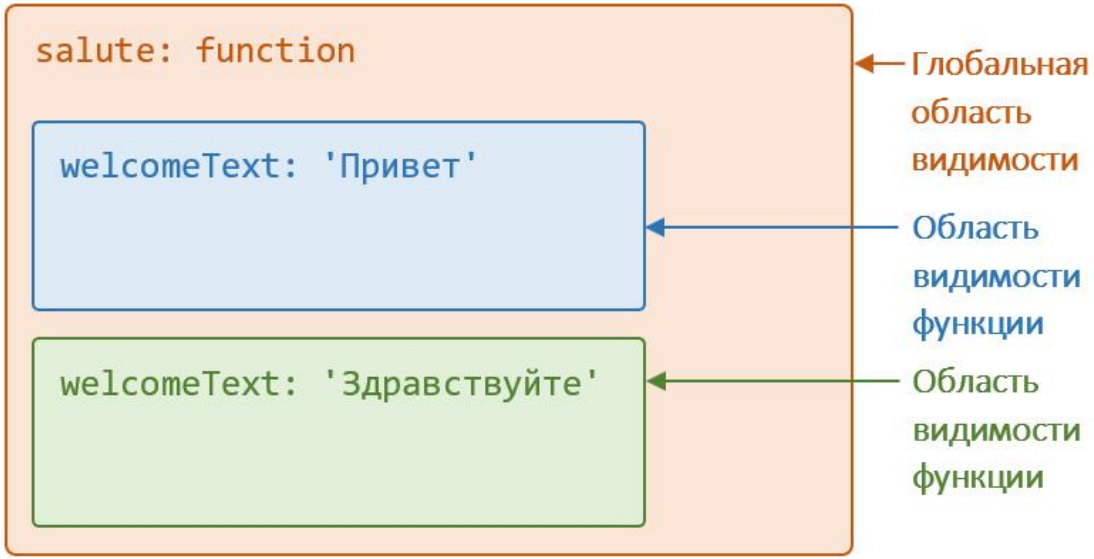
Таким образом, аргумент функции недоступен вне этой функции. Это произошло, поскольку аргумент функции является **локальной переменной**. Он существует только во время выполнения функции и доступен только внутри неё. Также говорят, что аргумент функции находится в **локальной области видимости** функции.

Кроме локальной существует **глобальная область видимости**. Переменные, созданные вне функций, то есть в основном коде программы, находятся в глобальной области видимости. Это означает, что к ним можно получить доступ в любой части программы.

THEORY

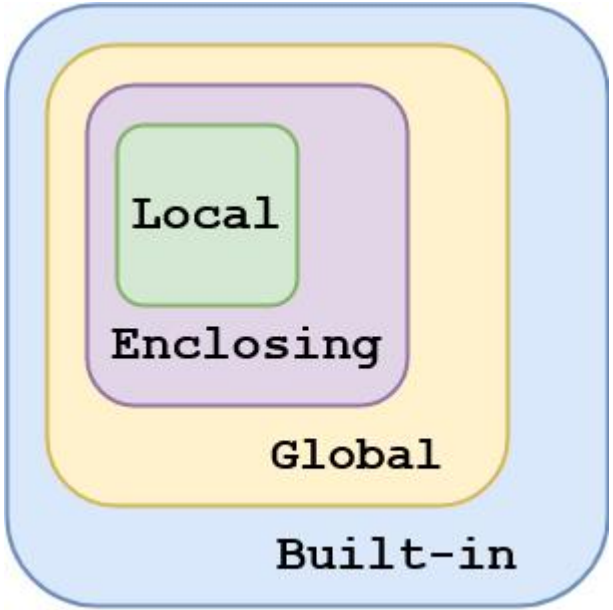
Для определения последовательности, в которой Python ищет значение переменной, используется правило LEGB. Это правило представляет собой порядок поиска переменных в следующих областях видимости: Local (локальная), Enclosing (вложенная), Global (глобальная) и Built-in (встроенная).

Области видимости



Порядок

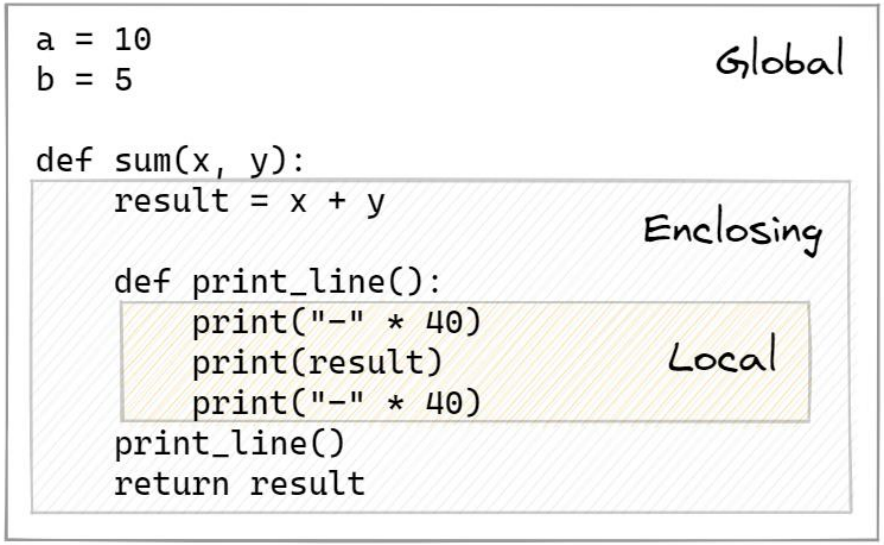
- LEGB
  - Local (локальная),
  - Enclosing (вложенная),
  - Global (глобальная)
  - Built-in (встроенная).



Порядок

- LEGB
  - Local (локальная),
  - Enclosing (вложенная),
  - Global (глобальная)
  - Built-in (встроенная).

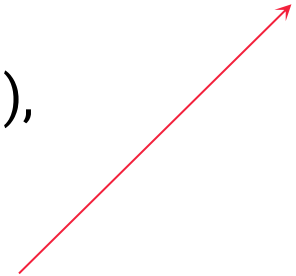
Предопределенные функции, типы и исключения, которые доступны в любой части программы



Порядок

- LEGB
  - Local (локальная),
  - Enclosing (вложенная),
  - Global (глобальная)
  - Built-in (встроенная).

Предопределенные функции, типы и исключения, которые доступны в любой части программы



1. `print()`
2. `len()`
3. `type()`
4. `input()`
5. `int(), float(), str()`
6. `range()`
7. `sum()`
8. `min(), max()`
9. `sorted()`

THEORY

Локальная область видимости: Переменные, созданные внутри функции, существуют в локальной области видимости этой функции и доступны только внутри неё.

Область видимости вложенных функций: Если функция определена внутри другой функции, её переменные доступны только внутри этой вложенной функции.

Глобальная область видимости: Переменные, определённые на уровне скрипта или модуля, считаются глобальными и доступны из любой части кода в том же модуле.

Встроенная область видимости: Это специальная область видимости, которая включает в себя все встроенные объекты и функции Python, доступные по умолчанию (например, print() и len()).

Упражнение

- LEGB
  - Local (локальная),
  - Enclosing (вложенная),
  - Global (глобальная)
  - Built-in (встроенная).

Предопределенные функции, типы и исключения, которые доступны в любой части программы

```
x = "global"

def outer():
    y = "outer local"
    def inner():
        z = "inner local"
        print(x)
        print(y)
        print(z)
    inner()

outer()
```

Упражнение

- LEGB
  - Local (локальная),
  - Enclosing (вложенная),
  - Global (глобальная)
  - Built-in (встроенная).

Предопределенные функции, типы и исключения, которые доступны в любой части программы

```
x = "global"

def outer():
    x = "outer local"
    def inner():
        x = "inner local"
        print(x)
    inner()

outer()
```

Ответ: inner local

- LEGB
  - Local (локальная),
  - Enclosing (вложенная),
  - Global (глобальная)
  - Built-in (встроенная).

Предопределенные функции, типы и исключения, которые доступны в любой части программы

```
x = "global"

def outer():
    x = "outer local"
    def inner():
        x = "inner local"
        print(x)
    inner()

outer()
```

Упражнение

- LEGB
  - Local (локальная),
  - Enclosing (вложенная),
  - Global (глобальная)
  - Built-in (встроенная).

Предопределенные функции, типы и исключения, которые доступны в любой части программы

```
x = "global"

def outer():
    x = "outer local"
    def inner():
        z = "inner local"
        print(x)
    inner()

outer()
```

THEORY

Замыкания часто используются в **функциональных языках программирования**.

Python их поддерживает, замыкания в нём могут быть очень полезными, так как они, например, позволяют создавать декораторы на основе функций.

```c
#include <stdio.h>

int main(void) {
  int x[10] = {1,4,34,53,1,92,4,8,2,44};
  int min = 99999;
  int max = -99999;

  for(int i = 0; i<10; i++){
    if (x[i] < min)
      min = x[i];
  }

  for(int i = 0; i<10; i++){
    if (x[i] > max)
      max = x[i];
  }

  printf("max: %d\tmin: %d\n",max,min);

  return 0;
}
```

Глобальная область видимости
доступны все функции из файла stdio.h

Локальная область видимости:
переменные x, min и max видны везде ниже их объявления

Область видимости переменной i
за пределами тела цикла переменная i не существует

Область видимости переменной i
это новая переменная i, она никак не связана с той, что выше

Сюжет 2

А как быть с не_константой?

Что будет на экране?

А

```
value = 42
changed = value
changed += 8
print(value)
```

Б

```
value = [42]
changed = value
changed[0] += 8
print(value)
```

В

```
def change():
  value += 8

value = 42
change()
print(value)
```

Г

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Д

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

Что будет на экране?

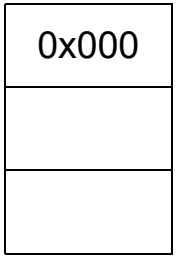
```
value = 42
changed = value
changed += 8
print(value)
```

А

Стек
value
changed

```
value = [42]
changed = value
changed[0] += 8
print(value)
```

Б




```
def change():
  value += 8

value = 42
change()
print(value)
```

В

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Г

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

Д

Что будет на экране?

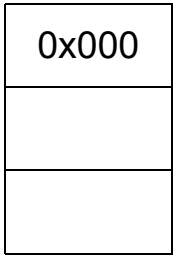
```
value = 42
changed = value
changed += 8
print(value)
```

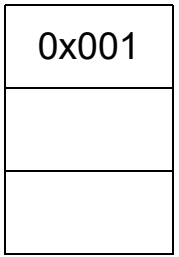
42

Стек
value
changed

Б

```
value = [42]
changed = value
changed[0] += 8
print(value)
```





В

```
def change():
  value += 8

value = 42
change()
print(value)
```

Г

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Д

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

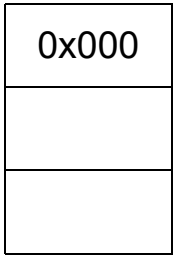
Что будет на экране?

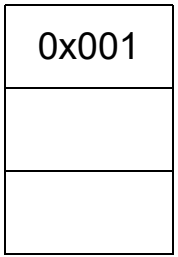
```
value = 42
changed = value
changed += 8
print(value)
```

42

```
value = [42]
changed = value
changed[0] += 8
print(value)
```

Б





Стек
value
changed

В

```
def change():
  value += 8

value = 42
change()
print(value)
```

Г

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Д

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

Что будет на экране?

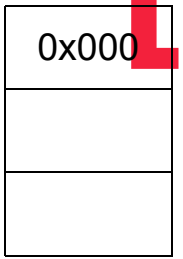
```
value = 42
changed = value
changed += 8
print(value)
```

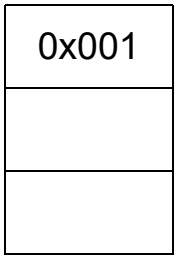
42

```
value = [42]
changed = value
changed[0] += 8
print(value)
```

[50]

Стек
value
changed





В

```
def change():
  value += 8

value = 42
change()
print(value)
```

Д

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Г

Что будет на экране?

```
value = 42
changed = value
changed += 8
print(value)
```

42

```
value = [42]
changed = value[:]
changed[0] += 8
print(value)
```

[42]

В

```
def change():
  value += 8

value = 42
change()
print(value)
```

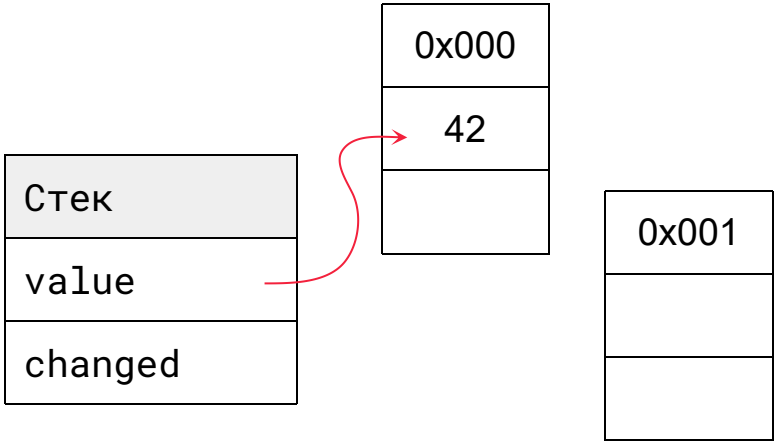
Д

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Г

Что будет на экране?



В

```
def change():
  value += 8

value = 42
change()
print(value)
```

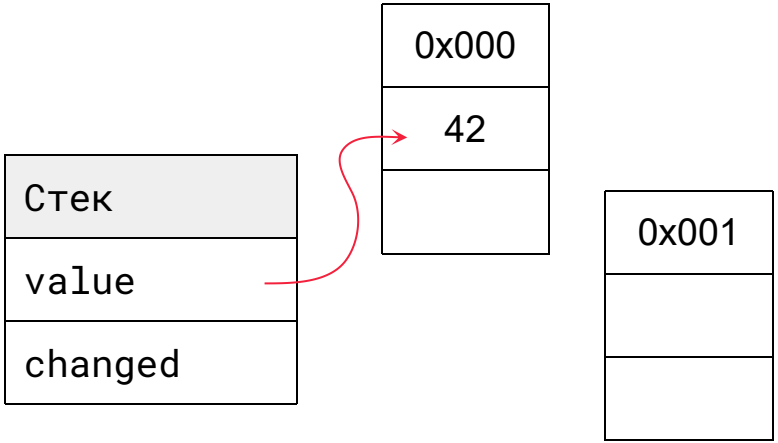
Д

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Г

Что будет на экране?



В локальном контексте нельзя изменять переменные неизменяемых типов

В

```
def change():
  value += 8

value = 42
change()
print(value)
```

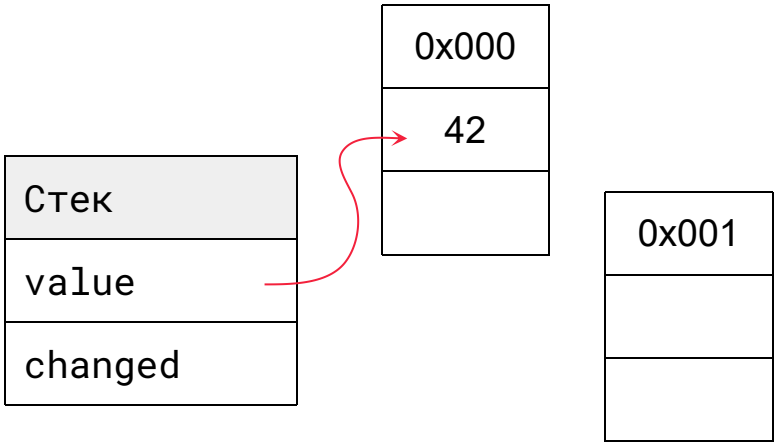
Д

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Г

Что будет на экране?



В локальном контексте нельзя изменять переменные неизменяемых типов

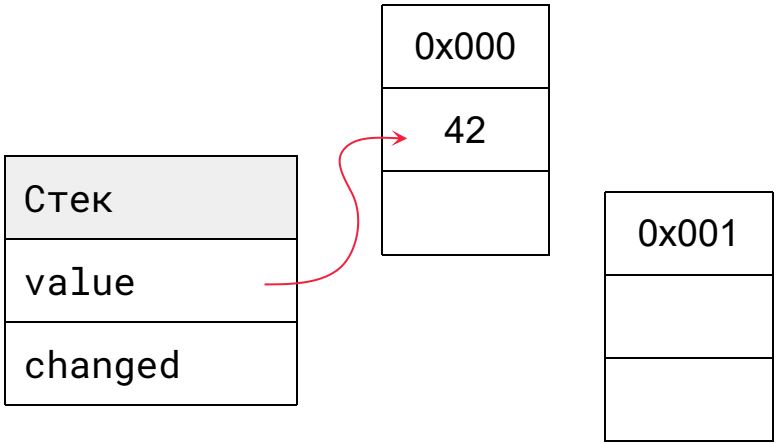
ERROR

```
def change():
  value += 8

value = 42
change()
print(value)
```

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Что будет на экране?



В локальном контексте можно изменять переменные изменяемых типов

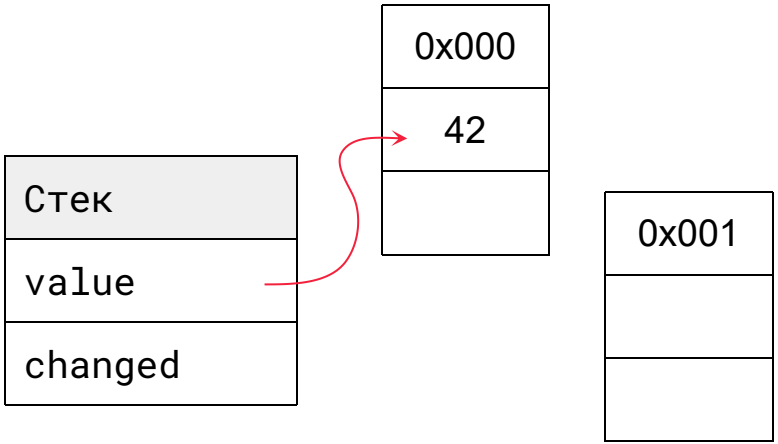
ERROR

```
def change():
  value += 8

value = 42
change()
print(value)
```

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Что будет на экране?



В локальном контексте можно изменять переменные изменяемых типов

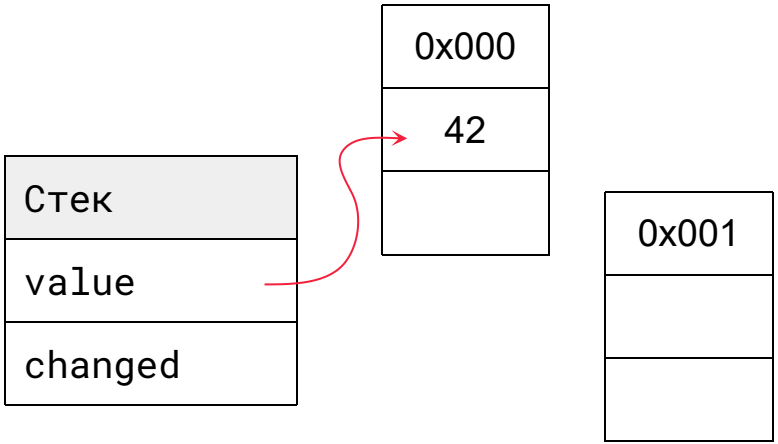
ERROR [50]

```
def change():
  value += 8

value = 42
change()
print(value)
```

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

Что будет на экране?



В локальном контексте **можно** изменять переменные изменяемых типов

50

```
def change():
  global value
  value += 8

value = 42
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

[50]

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

Что будет на экране?

```
def outer():
    n = 5
    def inner():
        n = 25
        print(n)
    inner() # 25
    print(n)# 5
outer()
```

50

```
def change():
  global value
  value += 8

value = 42
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

[50]

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

Что будет на экране?

```
def outer():
    n = 5
    def inner():
        nonlocal n
        n = 25
        print(n)
    inner() # 25
    print(n)# 25
outer()
```

50

```
def change():
  global value
  value += 8

value = 42
change()
print(value)
```

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

[50]

```
def change():
  value[0] += 8

value = [42]
change()
print(value)
```

Что будет на экране?

```
def change(value):
  value += 8

value = 42
change(value)
print(value)
```

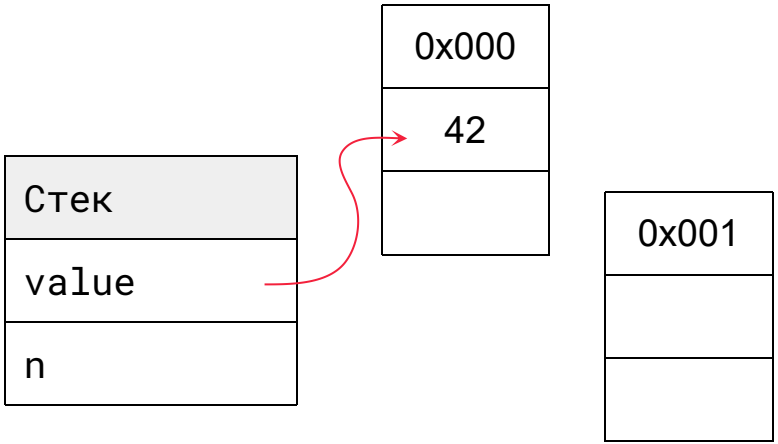
Что будет на экране?

Стек
value
n

0x000
42

```
def change(n):
  n += 8

value = 42
change(value)
print(value)
```

Что будет на экране?




```
def change(n):
    print(id(n))
    n += 8
    print(id(n))


value = 42
print(id(value))
print(change(value))
```

```
def change(n):
  n += 8

value = 42
change(value)
print(value)
```

Резюме

- Область видимости: LEGB
- Внутри функции можно видеть **глобальную переменную**
- Внутри функции можно менять **глобальную изменяемую** переменную
- Внутри функции можно менять **глобальную неизменяемую** переменную **только** при добавлении `global`
- Обратное `nonlocal`

Сюжет 3

Функция как объект

Боярский диалект языка С

- https://habr.com/ru/articles/41561/

```
использовати площадь какобычно амин

наместе двояко провѣрятичегоглаголют молчаливо
кагбе
    ѣжѣли получалка.сломалася молчаливо тогдауж
    кагбе
      молвити "Не лепо молвишь, барин!" амин
      возвѣрнути нуль спасихоспади1
    ага
    возвѣрнути один амин
ага
```

Боярский диалект языка Python

```
print("Hello, world!")
```

Боярский диалект языка Python

```
print("Благо, земляне!")
```

```
молви("Благо, земляне!")
```

Боярский диалект языка Python

```
def молви(txt):
  print(txt)



молви("Благо, земляне!")
```

Боярский диалект языка Python

```
def молви(txt):
  print(txt)
```

```
молви("Благо, земляне!", end="\n\n")
```

Боярский диалект языка Python

```
молви = print



молви("Благо, земляне!", end="\n\n")
```

Принцип

```
import this
```

Всё в Python является объектами:

```
>>> isinstance(1, object)
True
>>> isinstance(list(), object)
True
>>> isinstance(True, object)
True
>>> def foo():
...     pass
...
>>> isinstance(foo, object)
True
```

Принцип

```
import this
```

Всё в Python является объектами:

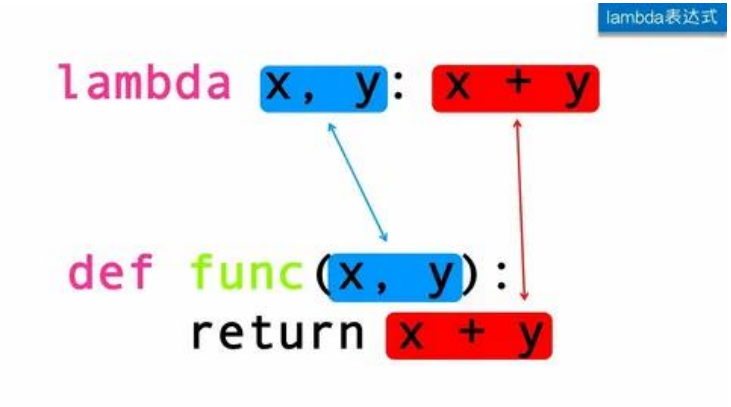
Каждый объект содержит как минимум три вида данных:

- Счётчик ссылок
- Тип
- Значение

Лямбда-функции

```
def defined_cube(y):
    return y * y * y
```

```
lambda_cube = lambda y: y*y*y
print(defined_cube(2))
print(lambda_cube(2))
```



Лямбда-функции

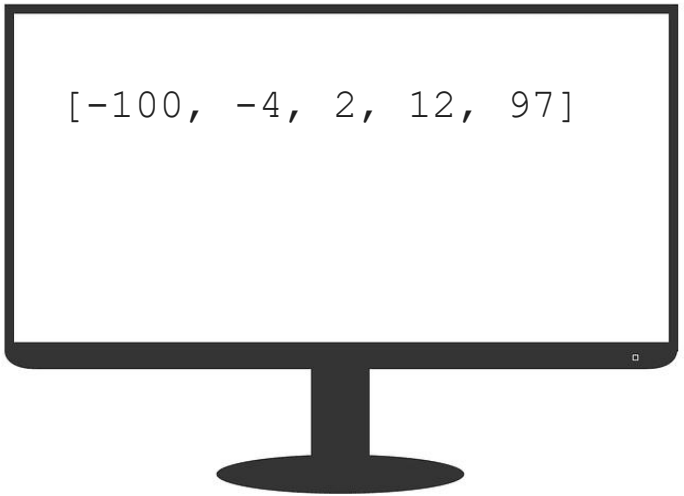
```
def defined_nesumma(x, y):
    return x + y - 1


lambda_nesumma = lambda x, y: x + y - 1
print(defined_nesumma(2, 4))
print(lambda_nesumma(2, 4))
```

Функция высшего порядка

```
numbers = [-100, 12, 2, 97, -4]
numbers.sort()
print(numbers)
```

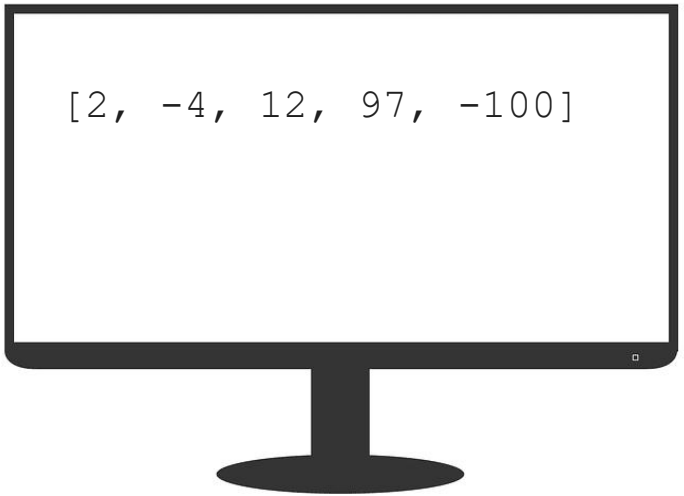
Функция высшего порядка

```
numbers = [-100, 12, 2, 97, -4]
numbers.sort()
print(numbers)
```



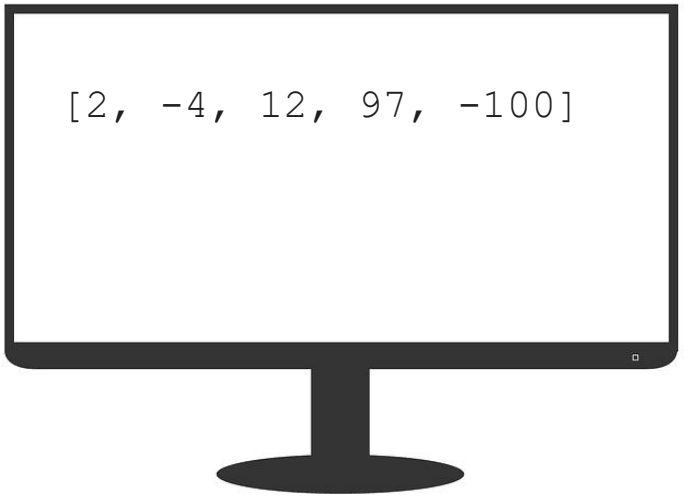
Функция высшего порядка

```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(???)
print(numbers)
```



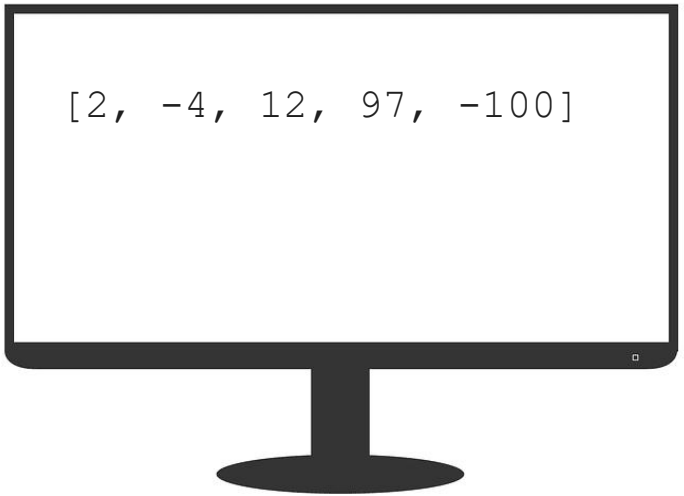
Функция высшего порядка

```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=comparator)
print(numbers)
```



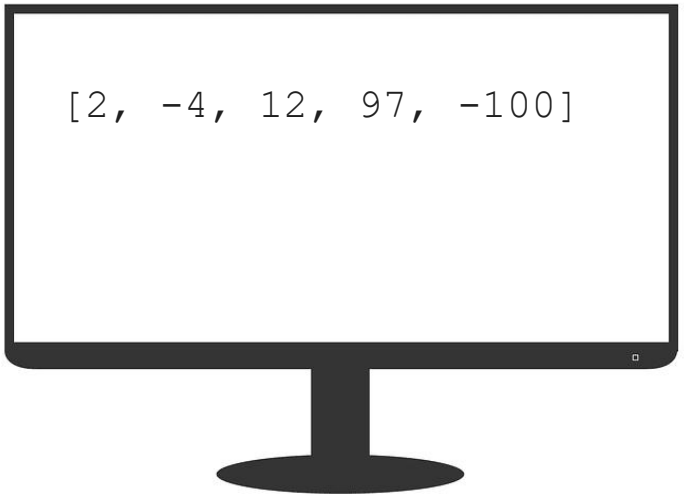
Функция высшего порядка

```
def comparator(x):
    return abs(x)


numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=comparator)
print(numbers)
```



Функция высшего порядка

```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: abs(x))
print(numbers)
```



Функция высшего порядка

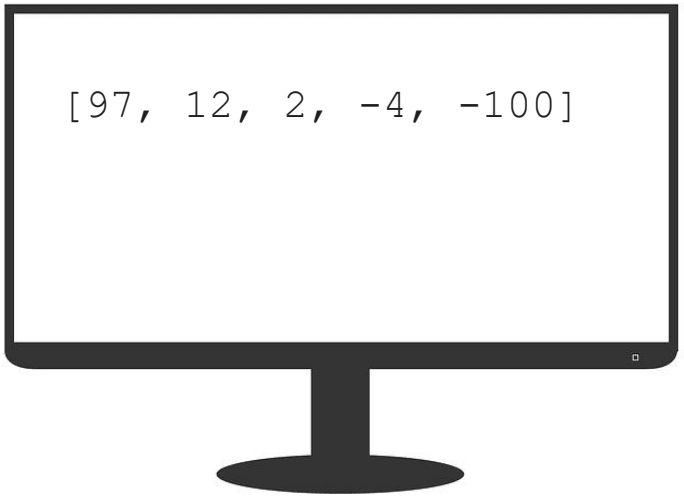
```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: x**2)
print(numbers)
```

Функция высшего порядка

```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: -x)
print(numbers)
```

Функция высшего порядка

```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: -x)
print(numbers)
```



Функция высшего порядка

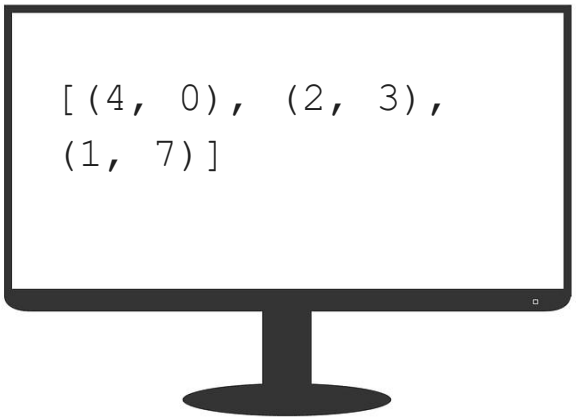
```
numbers = [[1, 7], [2, 3], [4, 0]]
numbers.sort(key=lambda x: x[0]+x[1])
print(numbers)
```

Функция высшего порядка

```
numbers = [(1, 7), (2, 3), (4, 0)]
numbers.sort(key=lambda x: x[0]+x[1])
print(numbers)
```

Функция высшего порядка

```
numbers = [(1, 7), (2, 3), (4, 0)]
numbers.sort(key=lambda x: x[0]+x[1])
print(numbers)
```



Сюжет 4

Функции высшего порядка

map

```
zoo = ['cat', 'hippo', 'dog']
numbers = []
for i in range(len(zoo)):
        numbers.append(zoo[i]
[0])
```



```
zoo = ['cat', 'hippo', 'dog']
numbers = []
for _x in zoo:
        numbers.append(_x[0])
```



```
zoo = ['cat', 'hippo', 'dog']
numbers = [_x[0] for _x in zoo]
```

map

```
zoo = ['cat', 'hippo', 'dog']
numbers = []
for i in range(len(zoo)):
        numbers.append(zoo[i]
[0])
```

```
zoo = ['cat', 'hippo', 'dog']
numbers = []
for _x in zoo:
        numbers.append(_x[0])
```

```
zoo = ['cat', 'hippo', 'dog']
numbers = [_x[0] for _x in zoo]
```

```
zoo = ['cat', 'hippo', 'dog']
numbers = map(lambda x: x[0], zoo)
```

map

```
zoo = ['cat', 'hippo', 'dog']
numbers = []
for i in range(len(zoo)):
        numbers.append(zoo[i]
[0])
```

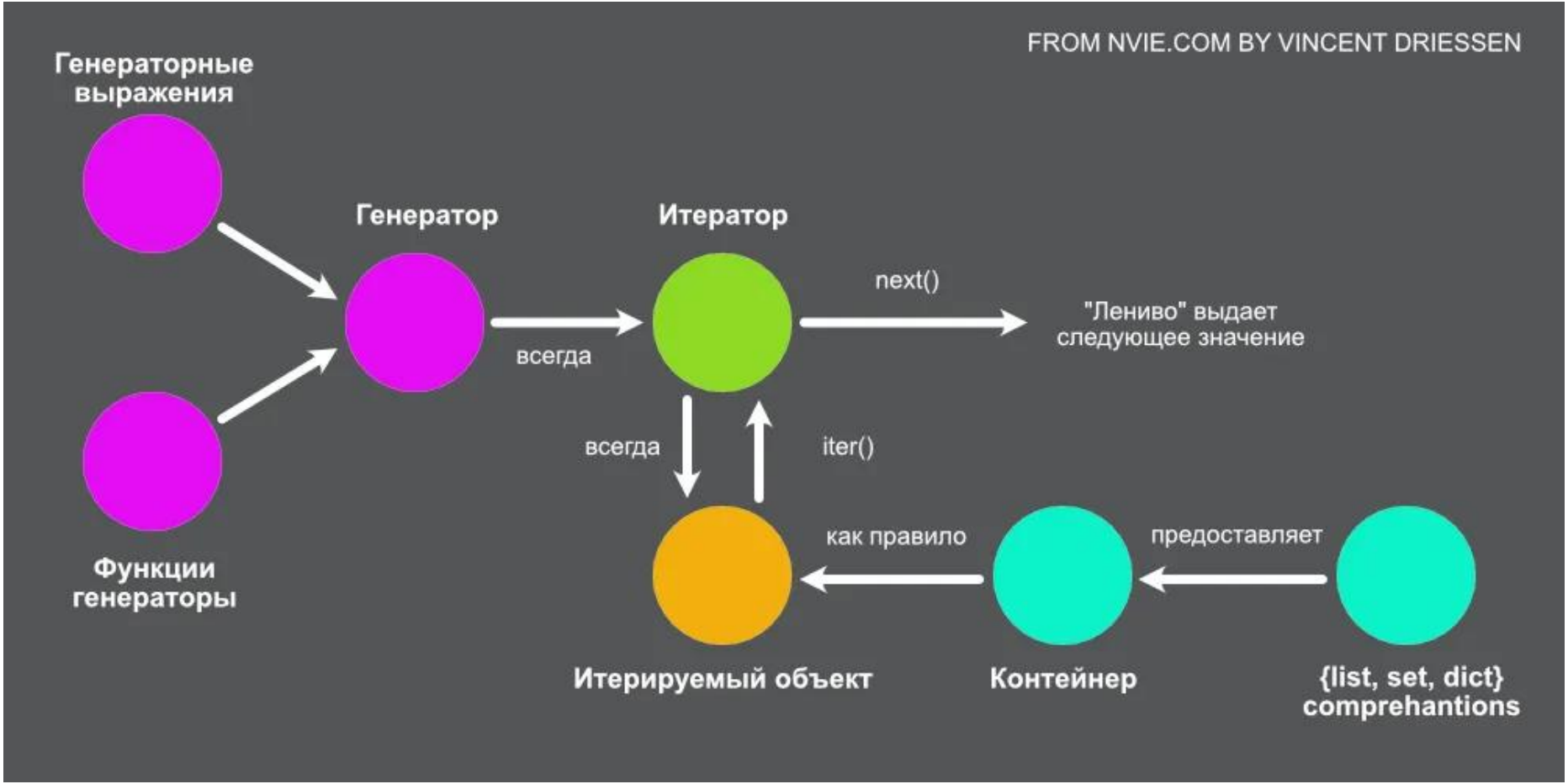
```
zoo = ['cat', 'hippo', 'dog']
numbers = []
for _x in zoo:
        numbers.append(_x[0])
```

```
zoo = ['cat', 'hippo', 'dog']
numbers = [_x[0] for _x in zoo]
```

```
zoo = ['cat', 'hippo', 'dog']
numbers = list(map(lambda x: x[0], zoo))
```

Итератор

```
zoo = ['cat', 'hippo', 'dog']
numbers = map(lambda x: x[0], zoo)


next(numbers)
```


FROM NVIE.COM BY VINCENT DRIESSEN
Генераторные выражения
Функции генераторы
Генератор
всегда
Итератор
next()
"Лениво" выдает следующее значение
всегда
iter()
Итерируемый объект
как правило
Контейнер
предоставляет
{list, set, dict} comprehantions

Фильтр

```
zoo = ['cat', 'hippo', 'dog']
result = filter(lambda x: len(x)>3, zoo)


list(result) # [hippo]
```

Сюжет 5

Вспомнить все

Замыкание (Closure)

```
def outer_func():
    name = "username"
    def inner_func():
        print(f"Hello, {name}!")
    inner_func()

outer_func()

greeter = outer_func()
print(greeter)
```

Вложенная функция

Замыкание (Closure)

```
def outer_func():
    name = "username"
    def inner_func():
        print(f"Hello, {name}!")
    inner_func()

outer_func() # Hello, username!

greeter = outer_func() # Hello, username!
print(greeter) # None
```

Вложенная функция

Замыкание (Closure)

- **Замыкание** — это функция, которая определена внутри другой функции и использует переменные из локальной области видимости внешней функции.

```
def outer_func():
    name = "username"
    def inner_func():
        print(f"Hello, {name}!")
    inner_func()

outer_func() # Hello, username!

greeter = outer_func()
print(greeter) # None
```

Вложенная функция

Замыкание (Closure)

- **Замыкание** — это функция, которая определена внутри другой функции и использует переменные из локальной области видимости внешней функции.

```
def outer_func():
    name = "username"
    def inner_func():
        print(f"Hello, {name}!")
    return inner_func


outer_func()
greeter = outer_func()
greeter() # Hello, username!
```

Вложенная функция

Замыкание (Closure)

```
def outer_func():
    name = "username"
    return lambda: print(f"Hello, {name}!")

greeter = outer_func()
greeter() # Hello, username!
```

Упражнение

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

counter = make_counter()
counter()
counter()
counter()
```

Упражнение

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

counter = make_counter()
counter() # 1
counter() # 2
counter() # 3
```

Декораторы

Декораторы — это способ изменить поведение функции или класса, обернув его в другую функцию или класс.

```
def uppercase(func):
    def wrapper():
        original_result = func()
        modified_result = original_result.upper()
        return modified_result
    return wrapper

@uppercase
def greet():
    return 'Hello!'

print(greet)
```

Декораторы

Декораторы — это способ изменить поведение функции или класса, обернув его в другую функцию или класс.

```
def uppercase(func):
    def wrapper():
        original_result = func()
        modified_result = original_result.upper()
        return modified_result
    return wrapper

@uppercase
def greet():
    return 'Hello!'

print(greet) # HELLO
```

Декораторы

Декораторы — это способ изменить поведение функции или класса, обернув его в другую функцию или класс.

```
def make_greeting_decorator(greeting):
    def decorator(func):
        def wrapper(name):
            return f"{greeting}, {func(name)}!"
        return wrapper
    return decorator


def greet(name):
    return name

print(greet("John"))  # John!
```

Декораторы

Декораторы — это способ изменить поведение функции или класса, обернув его в другую функцию или класс.

```
def make_greeting_decorator(greeting):
    def decorator(func):
        def wrapper(name):
            return f"{greeting}, {func(name)}!"
        return wrapper
    return decorator

@make_greeting_decorator("Hello")
def greet(name):
    return name

print(greet("John"))  # Hello, John!
```

Декораторы

Декораторы — это способ изменить поведение функции или класса, обернув его в другую функцию или класс.

```
def make_greeting_decorator(greeting):
    def decorator(func):
        def wrapper(name):
            return f"{greeting}, {func(name)}!"
        return wrapper
    return decorator

@make_greeting_decorator("Dear")
def greet(name):
    return name

print(greet("John"))  # Dear, John!
```

Декораторы

Декораторы — это способ изменить поведение функции или класса, обернув его в другую функцию или класс.

```
def make_greeting_decorator(greeting):
    def decorator(func):
        def wrapper(name):
            return f"{greeting}, {func(name)}!"
        return wrapper
    return decorator

@make_greeting_decorator("Hi")
def greet(name):
    return name

print(greet("John"))  # Hi, John!
```

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Опрос и дз

Итого

- Опрос в конце: https://otus.ru/polls/122212/

Занятие «Функции.Часть 2 (декораторы)»

Цели занятия

познакомиться с функциями высшего порядка;
познакомиться с областями видимости функций;
разобраться с замыканием функций;
разобрать принцип работы декораторов.

Свернуть

Краткое содержание

функции как полноценные объекты;
функции высшего порядка: map, filter;
области видимости, nonlocal и global;
замыкание функций;
декораторы;
декораторы с параметром.

Свернуть

Итого

- Опрос в конце: https://otus.ru/polls/122212/

```
def change(a, b):
   tmp = b
   b = a
   a = tmp

x, y = 2, 7
change(x, y)
print(x, y)
```

```
def change(a, b):
   return b, a

x, y = 2, 7
x, y = change(x, y)
print(x, y)
```

Следующий вебинар

Работа с файлами

- Ссылка на вебинар будет в ЛК за 15 минут
- Материалы к занятию в ЛК — можно изучать
- Обязательный материал обозначен красной лентой