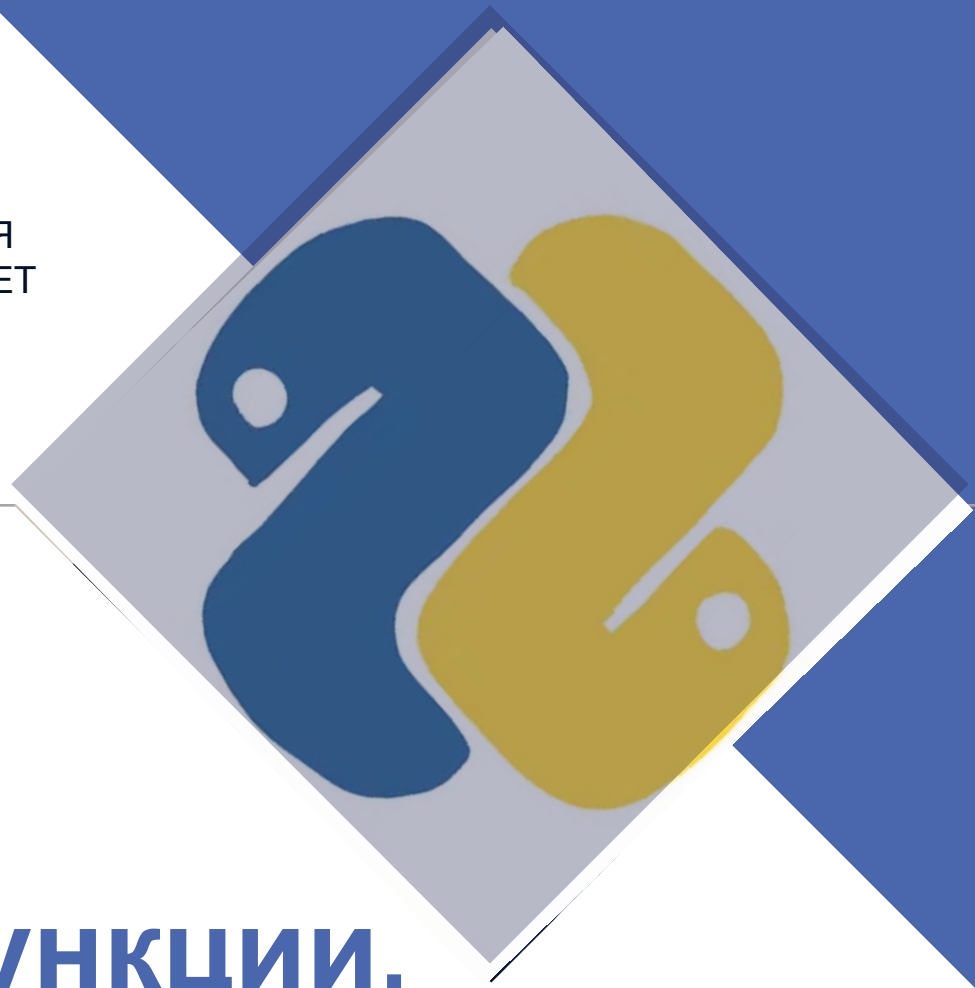
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАБАРДИНО-БАЛКАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. Х.М. БЕРБЕКОВА»

ИНСТИТУТ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И ЦИФРОВЫХ
ТЕХНОЛОГИЙ



Рекурсивные функции, анонимные функции, области видимости, вложенные функции

Работа с функциями

Функцию можно рассматривать как отдельный тип данных. То есть Python позволяет присвоить переменной какую-нибудь функцию и затем, используя переменную, вызывать ее.

```
def summ(a, b): # функция сложения
    return (a + b)

x = summ # переменной x присваиваем функцию sum

print(x(10, 20)) # вызов функции сложения
```

```
30
```

Работа с функциями

Переменные, которым присвоены функции (делегаты) могут быть переопределены, так же, как и другие переменные

```
def up(a): # определение функций
    return (a + 1)

def down(a):
    return (a - 1)

x = up # присваеваем переменной функцию
y = x
x = down # можно поменять присвоенную
функцию
print(x(10)) # вызов функций через
```

```
9
11
```

Работа с функциями

Естественно функцию можно не только присвоить переменной, но и передать другой функции в качестве аргумента

```
def doit (a, b, function): # функция принимает аргументы: два числа и другую функцию
    return function(a, b) # и использует переданную функцию

def plus(x, y): # функции сложения и вычитания
    return (x + y)
def minus(x, y):
    return (x - y)

print(doit(10, 5, plus)) # вызов doit,
```

```
15
5
```

Работа с функциями

Кроме того, в результате работы функции может быть возвращена другая функция

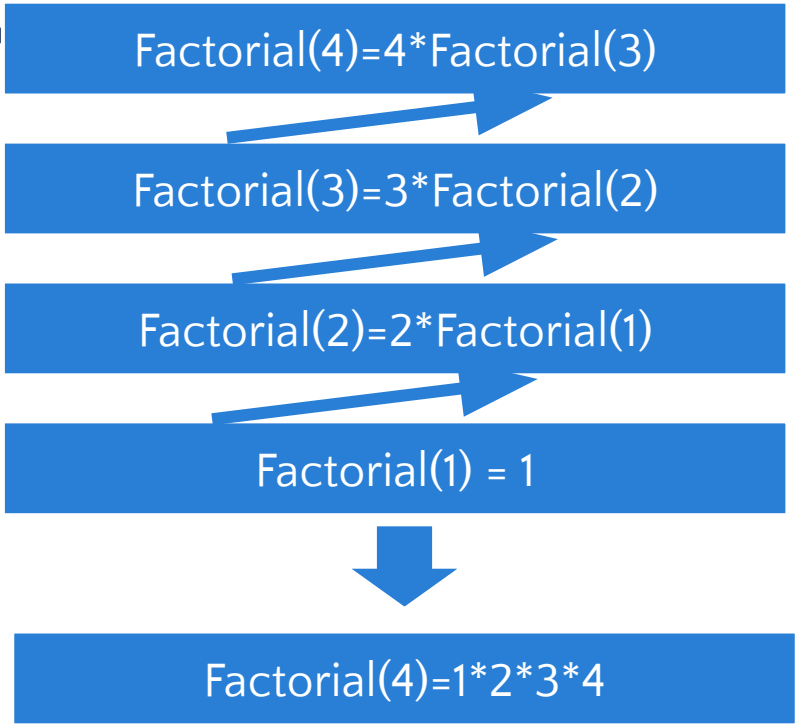
```
def plus(a, b):
    return (a + b)
def minus (a, b):
    return (a - b)
def operation(type):
    if (type == 1): # в зависимости от type
        return plus # функция возвращает одну из
двух функций
    else:
        return minus
func = operation(1) # в func записывается
результат работы operation
print(func(5, 3)) # вызовет plus(5, 3)
```

```
8
2
```

Рекурсия

Рекурсия – это возможность некоторого объекта или понятия быть частью самого себя.

В программировании рекурсия – это возможность из тела функции вызвать эту же функцию. Такая возможность иногда позволяет упростить код, но неправильное использование вызовет зацикливание программы



Рекурсия

Для использования рекурсии достаточно вызвать функции в ней же.

```
def func():
    print("text")
    func() # ссылка на саму функции в ее теле

func() # вызов рекурсивной функции
```

```
text
text
text
text
text
text
text
```

Рекурсия

Более правильный подход использовать условие для рекурсии. Например, расчет факториала числа N как N * (N – 1)! с учетом,

```
def factorial(n):
    if n == 1:
        return 1 # исключение при расчете 1
    else:
        return n * factorial(n - 1) # рекурсия

print(factorial(6))
```

```
720
```

Анонимные функции

Анонимной функцией (лямбда функции) – это те функции, которые не имеют уникального имени и объявляются в месте использования. Они используются для сокращения кода в случае, если функции не обязательно давать имя.

В python для анонимных функций используется ключевое слово lambda

lambda [параметр, параметр …] **:** инструкция (или результат)

Анонимные функции

Анонимную функцию можно хранить в переменной (делегате)

```
func = lambda: print("it is lambda") # запись анонимной функции в переменную

func() # использование переменной
```

```
it is lambda
```

Анонимные функции

Если анонимная функция имеет параметры, то они определяются после ключевого слова lambda. Если анонимная функция возвращает какой-то результат, то он указывается после двоеточия.

```
plus = lambda a, b: a + b # функция принимает
числа a и b и возвращает (a + b)

print(plus(10, 20))
print(plus(30, 40))
```

```
30
70
```

Анонимные функции

Если ваша функция принимает функцию в качестве параметра или возвращает ее в качестве результата – то в таком случае тоже можно использовать анонимные функции.

```
def doit(a, b, operation): # doit ожидает функцию
на вход
    print(operation(a, b))

doit(2, 3, lambda x, y: x * y) # в doit передается
анонимная функция
```

```
30
70
```

Анонимные функции

Кроме того, анонимные функции могут использоваться как аргумент базовых функций python. Например, функция map позволяет обработать элементы списка и должна получать функцию обработчик и сам список.

```
data = [1, 2, 3, 4, 5] # список
result = map(lambda x: x**2, data) # в map передается функция возведения в квадрат
print(list(result))
```

```
[1, 4, 9, 16, 25]
```

Вложенные функции

Вложенные функции (внутренние функции) – функции, которые определены внутри других функций.
В Python такая функция может иметь доступ к переменным и именам, определенным во включающей функции

```
def [имя функции] ( [аргументы] ):
    [тело функции]
    def [имя вложенной функции] ( [аргументы] ):
        [тело вложенной функции]
```

Вложенные функции

Внутренняя функция может быть вызвана во внешней, но не может использоваться за ее пределами. Это позволит скрыть те функции, которые не нужны за пределами внешней функции.

```
def outer(): # внешняя функция
    print("it is outer")
    def inner(): # внутренняя функция
        print("it is inner")

    inner() # вызов внутренней функции

outer() # запуск внешней функции
# inner() # внутреннюю функцию здесь запустить не выйдет
```

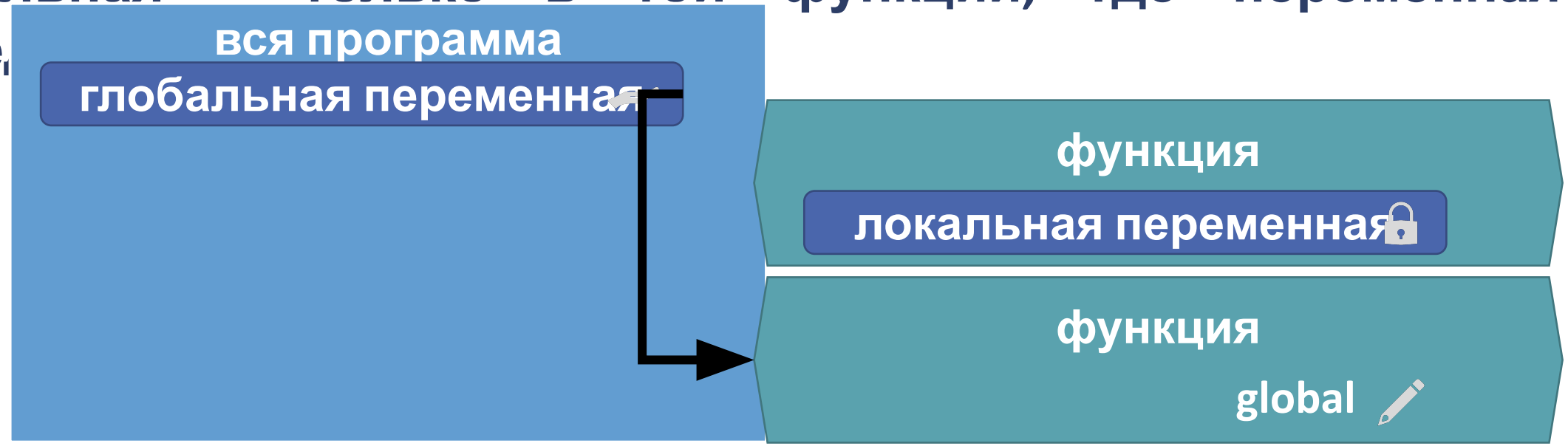
```
it is
outer
it is
inner
```

Область видимости

Область видимости переменной – эта та область программы (те функции), где можно использовать данную переменную.

Выделяют две основные области видимости:

- **глобальная – переменная доступна во всей программе (и в ее функциях)**
- **локальная – только в той функции, где переменная опре**



Область видимости

Если переменная определена вне функций – она глобальная и доступна во всех функциях

```
name = "Ezio" # глобальная переменная, определена
вне функций

def Hi():
    print("Hi", name) # функция использует
глобальную переменную
def Hello():
    print("Hello", name)

Hi() # вызов функций
Hello()
```

```
Hi Ezio
Hello Ezio
```

Область видимости

Локальная переменная доступна только в той функции, в которой она определена

```
def Hi():
    name = "Ezio"  # локальная переменная функции Hi
    print("Hi", name) # функция использует локальную переменную
def Hello():
    name = "Altair" # дрругая локальная переменная (имя переменной может повторяться в разных контекстах)
    print("Hello", name)

Hi() # вызов функций
Hello()
```

```
Hi Ezio
Hello Altair
```

Область видимости

Если в функции есть локальная переменная с таким же именем, что и у глобальной – то в рамках данной функции будет использоваться только локальная переменная (вместо глобальной)

```
name = "Altair" # глобальная переменная

def Hi():
    name = "Ezio"  # локальная переменная скрывает глобальную
    print("Hi", name) # функция использует локальную переменную
def Hello():
    print("Hello", name) # а здесь используется глобальная

Hi() # вызов функций
Hello()
print(name)
```

```
Hi Ezio
Hello
Altair
```

Область видимости

Ключевое слово global в функции позволяет обращаться к глобальной переменной и менять ее значение

```
name = "Altair" # глобальная переменная
def Hi():
    global name # обращаемся к глобальной
переменной
    name = "Ezio"  # можем изменить значение
глобальной переменной
    print("Hi", name)
def Hello():
    print("Hello", name) # а здесь используется
глобальная

Hello() # имя еще не изменили
Hi() # функция HI меняет глобальную переменную с
```

```
Hello
Altair
Hi Ezio
Hello Ezio
```

Область видимости

При этом у вложенных функций могут быть свои локальные переменные

```
temp = 0 # глобальная переменная
def outer():
    temp = 23 # локальная (для outer)

    def inner():
        temp = -10 # локальная (для inner)
        print(temp)
    inner()
    print(temp)

outer() # печать двух разных локальных temp
print(temp) # и печать глобальной temp
```

```
-10
23
0
```

Область видимости

Если во вложенной функции необходимо использовать локальную переменную внешней функции – используется

```
temp = 0 # глобальная переменная
def outer():
    temp = 23 # локальная (для outer)

    def inner():
        nonlocal temp # будет использоваться
локальная переменная внешней функции
        temp = -10 # изменяем внешнюю локальную
переменную
        print(temp)
    inner()
    print(temp)
outer() # печать двух локальных переменных
```

```
-10
-10
0
```

Замыкания

Замыкания – это функции, которая запоминает свое окружение (состояние внешней функции) даже если она выполняется вне своей области видимости.

Для этого необходимо:

- **внешняя функция в которой определены переменные (окружение)**
- **вложенная функция, которая использует это окружение**
- **внешняя функция возвращает вложенную**

Замыкания

Например, функция inner запоминает состояние внешнего окружения (переменную n функции power) и использует в

```
def power(n): # внешняя функция с переменной n
    def inner(m): # внутренняя функция
        return n ** m # использование окружения
(n)
    return inner # power возвращает внутреннюю
функцию

fn = power(10) # fn = inner()
print(fn(3)) # возведение 10 в степени разных
чисел
print(fn(4))
print(fn(5))
```

```
1000
10000
100000
```

Декораторы

Декораторы - функция, которая в качестве параметра получает функцию и в качестве результата также возвращает функцию. Декоратор позволяет модифицировать исходную функцию без явного изменения исходного кода функции.

```
def [имя декоратора] ( [имя функции] ):
    [тело декоратора]
    return [модифицированная функция]

@ [имя декоратора]
def [имя исходной функции] ( [аргументы] ):
    [тело исходной функции]
```

Декораторы

Например, декоратор может добавить к исходной функции дополнительный функционал

```
def select(input_func): # определение функции декоратора
    def output_func():  # определяем функцию, которая будет выполняться вместо оригинальной
        print("Hello")  # модификация оригинальной функции
        input_func()  # вызов оригинальной функции

    return output_func  # возвращаем новую функцию

@select  # применение декоратора select
def hello(): # определение оригинальной функции
    print("World") # содержимое оригинальной
```

```
Hello
World
```

ПРАКТИКА

Пример 1

Необходимо создать функцию setFuncs, принимающую список чисел и две другие функции в качестве аргумента. Функция должна применять к четным первую функцию-аргумент, а к нечетным - вторую и выводить результат. Проверьте работу созданной функции передав ей список чисел и функции возведение в квадрат и возведение в куб. Функции надо передавать в виде лямбда функций.

```
data = [1, 2, 3, 4, 5]
setFuncs(data, lambda a: a**2, lambda a: a**3)
```

```
1
4
27
16
125
```

Пример 1

```
def setFuncs(data, firstFunc, secondFunc):
    for i in data:
        if (i%2==0):
            print(firstFunc(i))
        else:
            print(secondFunc(i))

data = [1, 2, 3, 4, 5]
setFuncs(data, lambda a: a**2, lambda a: a**3)
```

Пример 2

Создайте функцию story для вывода всех имен из списка (который передается как аргумент). При создании функции вместо цикла нужно использовать рекурсию (то есть вызывать функцию из нее же).

```
story(["jack", "john", "james"])
```

```
hello, jack
hello, john
hello, james
```

Пример 2

```
def story(names):
    print("hello,", names[0])
    if (len(names)>1):
        story(names[1:])

story(["jack", "john", "james"])
```

Пример 3

Создайте функцию showStars, которая получает список чисел и выводит такое же количество звездочек в строку (через пробел), то есть при вводе 2 3 5 на выход должны вывестись ** * *****. Вывод указанного количества звездочек реализуйте в качестве внутренней функции.**

```
showStars([2, 4, 6, 8, 10])
```

```
** **** ****** ******** **********
```

Пример 3

```
def showStars(data):
    def stars(n):
        return (n * "*")
    text=""
    for i in data:
        text = text + stars(i) + "  "
    print(text)

showStars([2, 4, 6, 8, 10])
```

Пример 4

Создайте программу с глобальной переменной temp (температура в помещении) и двумя функциями для heat и cold (нагрев и охлаждение). Обе функции принимают значение времени (time) и имеют свои (различные) коэффициенты эффективности (coef), при этом первая увеличивает температуру на time*coef, а вторая уменьшает. Проверьте работу вашей программы.

```
print(temp)
heat(2)
print(temp)
cold(2)
print(temp)
```

```
20
24
14
```

Пример 4

```
temp=20
def heat(time):
    coef = 2
    global temp
    temp += coef * time
def cold(time):
    coef = 5
    global temp
    temp -= coef * time


print(temp)
heat(2)
print(temp)
cold(2)
print(temp)
```

Задача 1

Создайте функцию calc, которой можно передать символ + - * /, а функция должна вернуть соответствующую функцию, то есть при, вызове func("+") мы должны получить функцию, которая складывает два числа. Для проверки присвойте двум переменным результат созданной вами функции (например, для * и для /) и проверьте работу этих переменных на разных парах чисел.

```
mul = calc("*")
div = calc("/")
print(mul(10, 2))
print(div(10, 2))
```

```
20
5.0
```

Задача 1

```
def calc(operation):
    ??????????????


mul = calc("*")
div = calc("/")

print(mul(10, 2))
print(div(10, 2))
```

Задача 2

Необходимо создать функцию changeRange, которая принимает два числа (start и end) и функцию operation. В ней над всеми элементами ряда от start до end необходимо совершить operation. Функция должна вернуть сумму всех чисел от start до end, применив переданную ей операцию. Проверьте функцию, передав ей два числа и лямбда функцию (например, умножения на 10)

```
print(changeRange(1, 2, lambda a: a*10))
```

```
30
```

Задача 2

```
def changeRange(start, end, operation):
    ??????????????
    return (sum)

print(changeRange(1, 2, lambda a: a*10))
```

СПАСИБО ЗА ВНИМАНИЕ!