Л. р. 5 Работа с функциями в Python

Цель работы: Изучить следующие понятия: определение и вызов функций, синтаксис функций. Использование параметров функций. Использование оператора return. Использование значения None. Позиционные аргументы. Аргументы — ключевые слова. Значение параметра по умолчанию. Получение аргументов — ключевых слов с помощью **

Теоретическая часть

В предыдущих лабораторных работах все примеры кода представляли собой небольшие фрагменты. Они годятся для решения небольших задач, но никто не хочет набирать эти фрагменты раз за разом. Необходим какой-то способ организовать большой фрагмент кода в более удобные фрагменты. Первый шаг к повторному использованию кода — это создание функций.

Функция — это именованный фрагмент кода, отделенный от других. Она может принимает любое количество любых входных параметров и возвращать любое количество любых результатов. С функцией можно сделать две вещи:

• определить;

• вызвать.

Чтобы определить функцию, нужно написать def, имя функции, входные параметры, заключенные в скобки, и, наконец, двоеточие (:). Имена функций подчиняются тем же правилам, что и имена переменных (они должны начинаться с буквы или _ и содержать только буквы, цифры или _).

Для того, чтобы определить простейшую функцию, нужно использовать следующий формат:

```
>>> def add(x,y):
        return(x+y)
```

Инструкция
return говорит, что нужно вернуть значение. В нашем случае функция возвращает сумму x и y.

Теперь мы ее можем вызвать:

```
>>> add(1,10)
11
>>> add('abc', 'def')
'abcdef'
```

Функция может быть любой сложности и возвращать любые объекты (списки, кортежи, и даже функции!):

```
>>> def newfunc(n):
        def myfunc(x):
            return(x+n)
        return myfunc

>>> new=newfunc(100) #new - это функция
>>> new(200)
300
```

Функция может и не заканчиваться инструкцией return, при этом функция вернет значение None:

```
>>> def func():
        pass

>>> print(func())
None
```

Аргументы функции

Функция может принимать произвольное количество аргументов или не принимать их вовсе. Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными.

```
>>> def func(a, b, c=2): #c - необязательный аргумент
        return a+b+c

>>> func(1,2) #a=1, b=2, c=2(по умолчанию)
5
>>> func(1, 2, 3) #a=1, b=2, c=3
6
>>> func(a=1, b=3) #a=1, b=2, c=2
6
>>> func(a=3, c=6) #a=3, c=6, b не определён
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    func(a=3, c=6) #a=3, c=6, b не определён
TypeError: func() missing 1 required positional argument: 'b'
>>>
```

Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится *:

```
>>> def func(*args):
        return args

>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func()
()
>>> func(1)
(1,)
```

Как видно из примера, args - это кортеж из всех переданных аргументов функции, и с переменной можно работать также, как и с кортежем. Функция может принимать и произвольное число именованных аргументов, тогда перед именем ставится **:

```
>>> def func(**kwards):
        return kwards

>>> func(a=1, b=2, c=3)
{'a': 1, 'b': 2, 'c': 3}
>>> func()
{}
>>> func(a='python')
{'a': 'python'}
```

В переменной
kwargs у нас хранится словарь, с которым мы, опять-таки, можем делать все, что нам заблагорассудится.

Анонимные функции, инструкция lambda.

Анонимные функции могут содержать лишь одно выражение, но и выполняются они быстрее. Анонимные функции создаются с помощью инструкции lambda. Кроме этого, их не обязательно присваивать переменной, как делали мы инструкцией def func():

```
>>> func = lambda x,y: x+y
>>> func(1, 2)
3
>>> func('a', 'b')
'ab'
>>> (lambda x,y: x+y)(1, 2)
3
>>> (lambda x,y: x+y)('a', 'b')
'ab'
```

Lambda-функции, в отличие от обычной, не требуется инструкция return, а в остальном, ведет себя точно так же:

```
>>> func = lambda *args: args
>>> func(1, 2, 3, 4)
(1, 2, 3, 4)
```

Далее применение функций будет рассмотрено в практических заданиях.

Задания

Давайте действовать пошагово. Сначала определим и вызовем функцию, которая не имеет параметров. Перед вами пример простейшей функции:

```
>>> def do_nothing():
        pass
```

Даже если функции не нужны параметры, вам все равно необходимо указать круглые скобки и двоеточие в ее определении. Следующую строку необходимо выделить пробелами точно так же, как если бы это был оператор if. Python требует использовать выражение pass, чтобы показать, что функция ничего не делает. Это эквивалентно утверждению «Эта страница специально оставлена пустой» (несмотря на то что теперь это не так).

Функцию можно вызвать, просто написав ее имя и скобки. Она сработает, не сделав ничего:

```
>>> do_nothing()
>>>
```

Теперь определим и вызовем другую функцию, которая не имеет параметров и выводит на экран одно слово:

```
>>> def make_a_sound():
        print('quack')


>>> make_a_sound()
quack
```

Когда вы
вызываете функцию make_a_sound(), Python выполняет код, расположенный внутри ее описания. В этом случае он выводит одно слово и возвращает управление основной программе.

Попробуем написать функцию, которая не имеет параметров, но возвращает значение:

```
>>> def agree():
        return True
```

Вы можете вызвать эту функцию и проверить возвращаемое ею значение с помощью if:

```
>>> if agree():
        print('Отлично!')
else:
        print('Что-то пошло не так!')


Отлично!
```

Комбинация функций с проверками вроде if и циклами вроде while позволяет вам делать ранее недоступные вещи.

Теперь пришло время поместить аргументы в скобки после названия функции. Определим функцию echo(), имеющую один параметр anything. Она использует оператор return, чтобы отправить значение anything вызывающей стороне дважды, разделив их пробелом:

```
>>> def echo(anything):
        return anything + ' ' + anything
```

Теперь вызовем функцию echo(), передав ей строку 'Эгегей':

```
>>> echo('Эгегей')
'Эгегей Эгегей'
>>>
```

Значения, которые вы передаете в функцию при вызове, называются аргументами. Когда вы вызываете функцию с аргументами, значения этих аргументов копируются в соответствующие параметры внутри функций. В предыдущем примере функции echo() передавалась строка 'Эгегей'. Это значение копировалось внутри функции echo() в параметр anything, а затем возвращалось (в этом случае оно удваивалось и разделялось пробелом) вызывающей стороне.

Эти примеры функций довольно просты. Напишем функцию, которая принимает аргумент и что-то с ним делает. Мы адаптируем предыдущий фрагмент кода, который комментировал цвета. Назовем его commentary и сделаем так, чтобы он принимал в качестве аргумента строку color. Сделаем так, чтобы он возвращал описание строки вызывающей стороне, которая может решить, что с ним делать дальше:

```
>>> def commentary(color):
        if color=='красный':
                return "Это помидорка"
        elif color=='зеленый':
                return "Это огурчик"
        elif color=='ультрамарин':
                return "Я не знаю такого овоща"
        else:
                return "Я не знаю цвет " + color + "."


>>>
```

Вызовем функцию commentary(), передав ей в качестве аргумента строку 'blue' Функция сделает следующее: • присвоит значение 'blue' параметру функции color; • пройдет по логической цепочке if-elif-else;

• вернет строку;

• присвоит строку переменной comment.

Что мы получим в результате?

```
>>> commentary('голубой')
'Я не знаю цвет голубой.'
```

Функция может принимать любое количество аргументов (включая нуль) любого типа. Она может возвращать любое количество результатов (также включая нуль) любого типа. Если функция не вызывает return явно, вызывающая сторона получит результат None.

```
>>> print(do_nothing())
None
```

Использование значения None

None — это специальное значение в Python, которое заполняет собой пустое место, если функция ничего не возвращает. Оно не является булевым значением False, несмотря на то что похоже на него при проверке булевой переменной. Рассмотрим пример:

```
>>> thing = None
>>> if thing:
        print("It's some thing")
else:
        print("It's no thing")


It's no thing
```

Для того чтобы понять важность отличия None от булева значения False, используйте оператор is:

```
>>> if thing is None:
        print("It's nothing")
else:
        print("It's something")


It's nothing
```

Разница
кажется небольшой, однако она важна в Python. None потребуется вам, чтобы отличить отсутствующее значение от пустого. Помните, что целочисленные нули, нули с плавающей точкой, пустые строки (''), списки ([]), кортежи ((,)), словари ({}) и множества (set()) все равны False, но не равны None.

Напишем небольшую функцию, которая выводит на экран проверку на равенство None:

```
>>> def is_none(thing):
        if thing is None:
                print("It's None")
        elif thing:
                print("It's True")
        else:
                print("It's False")


>>>
```

Теперь выполним несколько проверок:

```
>>> is_none(None)
It's None
>>> is_none(True)
It's True
>>> is_none(False)
It's False
>>> is_none(0)
It's False
>>> is_none(0.0)
It's False
>>> is_none(())
It's False
>>> is_none([])
It's False
>>> is_none({})
It's False
>>> is_none(set())
It's False
```

Позиционные аргументы

Python довольно гибко обрабатывает аргументы функций в сравнении с многими языками программирования. Наиболее распространенный тип аргументов — это позиционные аргументы, чьи значения копируются в соответствующие параметры согласно порядку следования. Эта функция создает словарь из позиционных входных аргументов и возвращает его:

```
>>> def menu(wine, entree, dessert):
        return {'напиток': wine, 'основное блюдо': entree, 'десерт': dessert}

>>> menu('шардоне', 'цыплёнок табака', 'печеньки')
{'напиток': 'шардоне', 'основное блюдо': 'цыплёнок табака', 'десерт': 'печеньк
и'}
>>>
```

Несмотря на распространенность аргументов такого типа, у них есть недостаток, который заключается в том, что вам нужно запоминать значение каждой позиции. Если бы мы вызвали функцию menu(), передав в качестве последнего аргумента марку вина, обед вышел бы совершенно другим:

Аргументы — ключевые слова

```
>>> menu('говядина', 'мороженка', 'портвейн 777')
{'напиток': 'говядина', 'основное блюдо': 'мороженка', 'десерт': 'портвейн 777
'}
```

Для того чтобы избежать путаницы с позиционными аргументами, вы можете указать аргументы с помощью имен соответствующих параметров. Порядок следования аргументов в этом случае может быть иным:

```
>>> menu(entree='шашлык', dessert='пирожок с вишней', wine='кефир')
{'напиток': 'кефир', 'основное блюдо': 'шашлык', 'десерт': 'пирожок с вишней'}
```

Вы можете объединять позиционные аргументы и аргументы — ключевые слова. Сначала выберем вино, а для десерта и основного блюда используем аргументы — ключевые слова.

```
>>> menu('вдова Клико', dessert='тирамису', entree='рыба')
{'напиток': 'вдова Клико', 'основное блюдо': 'рыба', 'десерт': 'тирамису'}
```

Если вы вызываете функцию, имеющую как позиционные аргументы, так и аргументы — ключевые слова, то позиционные аргументы необходимо указывать первыми.

Указываем значение параметра по умолчанию

Вы можете указать значения по умолчанию для параметров. Значения по умолчанию используются в том случае, если вызывающая сторона не предоставила соответствующий аргумент. Эта приятная особенность может оказаться довольно полезной. Воспользуемся предыдущим примером:

```
def menu(wine, entree, dessert='вкусняшка'):
    return{'напиток': wine, 'основное блюдо' : entree, 'десерт' : dessert}
```

В этот раз мы вызовем функцию menu(), не передав ей аргумент dessert:

```
>>> menu('Кисель', 'Курица с нежным сливочным соусом')
{'напиток': 'Кисель', 'основное блюдо': 'Курица с нежным сливочным соусом', 'де
серт': 'вкусняшка'}
```

Если вы предоставите аргумент, он будет использован вместо аргумента по умолчанию:

```
>>> menu('Компот', 'Коржик', 'Карамелька')
{'напиток': 'Компот', 'основное блюдо': 'Коржик', 'десерт': 'Карамелька'}
```

Значение аргументов по умолчанию высчитывается, когда функция определяется, а не выполняется. Распространенной ошибкой новичков (и иногда не совсем новичков) является использование изменяемого типа данных вроде списка или словаря в качестве аргумента по умолчанию.

В следующей проверке ожидается, что функция buggy() будет каждый раз запускаться с новым пустым списком result, добавлять в него аргумент arg, а затем выводить на экран список, состоящий из одного элемента. Однако в этой функции есть баг: список будет пуст только при первом вызове. Во второй раз список result будет содержать элемент, оставшийся после предыдущего вызова:

```
>>> def buggy(arg, result=[]):
        result.append(arg)
        print(result)


>>> buggy('a')
['a']
>>> buggy('b') #ожидаем увидеть ['b']
['a', 'b']
```

Функция работала бы корректно, если бы код выглядел так:

```
>>> def works(arg):
        result = []
        result.append(arg)
        return result

>>> works('a')
['a']
>>> works('b')
['b']
```

Решить проблему можно, передав в функцию что-то еще, чтобы указать на то, что вызов является первым:

```
>>> def nonbuggy(arg, result=None):
        if result is None:
                result = []
        result.append(arg)
        print(result)


>>> nonbuggy('a')
['a']
>>> nonbuggy('b')
['b']
```

Получаем позиционные аргументы с помощью *

Если вы работали с языками программирования C или C++, то можете предположить, что астериск (*) в Python как-то относится к указателям. Это не так, Python не имеет указателей. Если символ * будет использован внутри функции с параметром, произвольное количество позиционных аргументов будет сгруппировано в кортеж. В следующем примере args является кортежем параметров, который был создан из аргументов, переданных в функцию print_args():

```
>>> def print_args(*args):
        print('Кортеж позиционных аргументов:', args)
```

Если вы вызовете функцию без аргументов, то получите пустой кортеж:

```
>>> print_args()
Кортеж позиционных аргументов: ()
```

Все аргументы, которые вы передадите, будут выведены на экран как кортеж args:

```
>>> print_args(3, 2, 1, 'Леген...', 'подождите...', '...дарно!!!')
Кортеж позиционных аргументов: (3, 2, 1, 'Леген...', 'подождите...', '...дарно!!!')
```

Это полезно при написании функций вроде print(), которые принимают произвольное количество аргументов. Если в вашей функции имеются также обязательные позиционные аргументы, *args отправится в конец списка и получит все остальные аргументы:

```
>>> def print_more(required1, required2, *args):
...     print('Need this one:', required1)
...     print('Need this one too:', required2)
...     print('All the rest:', args)
...
>>> print_more('cap', 'gloves', 'scarf', 'monocle', 'mustache wax')
Need this one: cap
Need this one too: gloves
All the rest: ('scarf', 'monocle', 'mustache wax')
```

При использовании * вам не нужно обязательно называть кортеж параметров args, однако это распространенная идиома в Python.

Получение аргументов — ключевых слов с помощью **

Вы можете использовать два астериска (**), чтобы сгруппировать аргументы — ключевые слова в словарь, где имена аргументов станут ключами, а их значения — соответствующими значениями в словаре. В следующем примере определяется функция print_kwargs(), в которой выводятся ее аргументы — ключевые слова:

```
>>> def print_kwargs(**kwargs):
...     print('Keyword arguments:', kwargs)
...
```

Теперь попробуйте вызвать ее, передав несколько аргументов:

```
>>> print_kwargs(wine='merlot', entree='mutton', dessert='macaroon')
Keyword arguments: {'dessert': 'macaroon', 'wine': 'merlot', 'entree': 'mutton'}
```

Внутри функции kwargs является словарем.

Если вы используете позиционные аргументы и аргументы — ключевые слова (*args и **kwargs), они должны следовать в этом же порядке. Как и в случае с args, вам не обязательно называть этот словарь kwargs, но это опять же является распространенной практикой.

Строки документации

Дзен Python гласит: удобочитаемость имеет значение. Вы можете прикрепить документацию к определению функции, включив строку в начало ее тела. Она называется строкой документации:

```
>>> def echo(anything):
...     'echo returns its input argument'
...     return anything
```

Вы можете сделать строку документации довольно длинной и даже, если хотите, применить к ней форматирование, что показано в следующем примере:

```
def print_if_true(thing, check):
    '''
    Prints the first argument if a second argument is true.
    The operation is:
    1. Check whether the *second* argument is true.
    2. If it is, print the *first* argument.
'''
if check:
    print(thing)
```

Для того чтобы вывести строку документации некоторой функции, вам следует вызвать функцию help(). Передайте ей имя функции, чтобы получить список всех аргументов и красиво отформатированную строку документации:

```
>>> help(echo)
Help on function echo in module __main__:
echo(anything)
    echo returns its input argument
```

Если вы хотите увидеть только строку документации без форматирования:

```
>>> print(echo.__doc__)
echo returns its input argument
```

Подозрительно выглядящая строка __doc__ является внутренним именем строки документации как переменной внутри функции. В пункте «Использование _ и __ в именах» в разделе «Пространства имен и область определения» данной работы объясняется причина появления всех этих нижних подчеркиваний.

Функции — это объекты первого класса

В Python объектами является все, включая числа, строки, кортежи, списки, словари и даже функции. Функции в Python являются объектами первого класса. Вы можете присвоить их переменным, использовать как аргументы для других функций и возвращать из функций. Это дает вам возможность решать с помощью Python такие задачи, справиться с которыми средствами многих других языков сложно, если не невозможно.

Для того чтобы убедиться в этом, определим простую функцию answer(), которая не имеет аргументов и просто выводит число 42:

```
>>> def answer():
...     print(42)
```

Вы знаете, что получите в качестве результата, если запустите эту функцию:

```
>>> answer()
42
```

Теперь определим еще одну функцию с именем run_something. Она имеет один аргумент, который называется func и представляет собой функцию, которую нужно запустить. Эта функция просто вызывает другую функцию:

```
>>> def run_something(func):
...     func()
```

Если мы передадим answer в функцию run_something(), то используем ее как данные, прямо как и другие объекты:

```
>>> run_something(answer)
42
```

Обратите внимание: вы передали строку answer, а не answer(). В Python круглые скобки означают «вызови эту функцию». Если скобок нет, Python относится к функции как к любому другому объекту. Это происходит потому, что, как и все остальное в Python, функция является объектом:

```
>>> type(run_something)
<class 'function'>
```

Попробуем запустить функцию с аргументами. Определим функцию add_args(), которая выводит на экран сумму двух числовых аргументов, arg1 и arg2:

```
>>> def add_args(arg1, arg2):
...     print(arg1 + arg2)
```

Чем является add_args()?

```
>>> type(add_args)
<class 'function'>
```

Теперь определим функцию, которая называется run_something_with_args() и принимает три аргумента:

• func — функция, которую нужно запустить;

• arg1 — первый аргумент функции func;

• arg2 — второй аргумент функции func:

```
>>> def run_something_with_args(func, arg1, arg2):
...     func(arg1, arg2)
```

Когда вы вызываете функцию run_something_with_args(), та функция, что передается вызывающей стороной, присваивается параметру func, а переменные arg1 и arg2 получают значения, которые следуют далее в списке аргументов. Вызов func(arg1, arg2) выполняет данную функцию с этими аргументами, потому что круглые скобки указывают Python сделать это.

Проверим функцию run_something_with_args(), передав ей имя функции add_args и аргументы 5 и 9:

```
>>> run_something_with_args(add_args, 5, 9)
14
```

Внутри функции run_something_with_args() аргумент add_args, представляющий собой имя функции, был присвоен параметру func, 5 — параметру arg1, а 9 — параметру arg2. В итоге получается следующая конструкция:

```
add_args(5, 9)
```

Вы можете объединить этот прием с использованием *args и **kwargs. Определим тестовую функцию, которая принимает любое количество позиционных аргументов, определяет их сумму с помощью функции sum() и возвращает ее: Функция sum() - это встроенная в Python функция, которая высчитывает сумму значений итерабельного числового (целочисленного или с плавающей точкой) аргумента. Мы определим новую функцию run_with_positional_args(), принимающую функцию и произвольное количество позиционных аргументов, которые нужно будет передать в нее:

```
>>> def run_with_positional_args(func, *args):
...     return func(*args)
```

Теперь вызовем ее:

```
>>> run_with_positional_args(sum_args, 1, 2, 3, 4)
10
```

Вы можете использовать функции как элементы списков, кортежей, множеств и словарей. Функции неизменяемы, поэтому вы можете даже применять их как ключи для словарей.

Вариант 0.

1. Определить, являются ли три треугольника равновеликими. Длины сторон вводить с клавиатуры. Для подсчёта площади треугольника использовать формулу Герона. Вычисление площади оформить в виде функции с тремя параметрами.

Формула Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

$$где \quad p = \frac{a+b+c}{2}$$

Решение:

```
import math
def s(x,y,z):
    p=(x+y+z)/2
    s=math.sqrt(p*(p-x)*(p-y)*(p-z))
    return s
A=[]
for i in range(3):
    print('Введите стороны ',i,'-го треугольника:')
    a=int(input('a:'))
    b=int(input('b:'))
    c=int(input('c:'))
    A.append(s(a,b,c))
for i in range(3):
    print('Площадь ',i,'-го треугольника {:.2f}'.format(A[i]))
if A[0]==A[1]:
    if A[0]==A[2]:
        print('Треугольники равновеликие')
else: print('Треугольники не равновеликие')
```

```
Введите стороны  0 -го треугольника:
a:3
b:4
c:5
Введите стороны  1 -го треугольника:
a:6
b:7
c:8
Введите стороны  2 -го треугольника:
a:9
b:10
c:11
Площадь  0 -го треугольника 6.00
Площадь  1 -го треугольника 20.33
Площадь  2 -го треугольника 42.43
Треугольники не равновеликие
```

2. Ввести одномерный массив A длиной m. Поменять в нём местами первый и последний элементы. Длину массива и его элементы ввести с клавиатуры. В программе описать процедуру для замены элементов массива. Вывести исходные и полученные массивы.

Решение:

```
def zam(X):
    tmp=X[0]
    X[0]=X[len(X)-1]
    X[len(X)-1]=tmp
A=[]
m=int(input('Введите длину массива:'))
for i in range(m):
    print('Введите ',i,'элемент массива')
    A.append(int(input()))
print(A)
zam(A)
print(A)
```

```
Введите длину массива:5
Введите  0 элемент массива
0
Введите  1 элемент массива
1
Введите  2 элемент массива
2
Введите  3 элемент массива
3
Введите  4 элемент массива
4
[0, 1, 2, 3, 4]
[4, 1, 2, 3, 0]
```

Вариант 1.

1. Составить программу для вычисления площади разных геометрических фигур.

2. Даны 3 различных массива целых чисел (размер каждого не превышает 15). В каждом массиве найти сумму элементов и среднеарифметическое значение.

Вариант 2.

1. Вычислить площадь правильного шестиугольника со стороной a, используя подпрограмму вычисления площади треугольника.

2. Пользователь вводит две стороны трех прямоугольников. Вывести их площади.

Вариант 3.

1.-Даны катеты двух прямоугольных треугольников. Написать функцию вычисления длины гипотенузы этих треугольников. Сравнить и вывести какая из гипотенуз больше, а какая меньше.

Преобразовать строку так, чтобы буквы каждого слова в ней были отсортированы по алфавиту.

Вариант 4.

1. Найти все простые натуральные числа, не превосходящие n, двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.

2. Задана окружность (x-a)2 + (y-b)2 = R2 и точки P(p1, p2), F(f1, f1), L(l1,l2). Выяснить и вывести на экран, сколько точек лежит внутри окружности. Проверку, лежит ли точка внутри окружности, оформить в виде функции.

Вариант 5.

1. Четыре точки заданы своими координатами X(x1, x2, x3), Y(y1, y2, y3), Z(z1, z2, z3), T(t1,t2, t3). Выяснить, какие из них находятся на минимальном расстоянии
друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде функции.

2. Напишите программу, которая выводит в одну строчку все делители переданного ей числа, разделяя их пробелами.

Вариант 6.

1. Составить программу для нахождения чисел из интервала [M, N], имеющих наибольшее количество делителей.

2. Составить программу вычисления площади выпуклого четырехугольника, заданного длинами четырех сторон и диагонали.

Вариант 7.

1. Даны числа X, Y, Z, T — длины сторон четырехугольника. Вычислить его площадь, если угол между сторонами длиной X и У — прямой. Использовать две подпрограммы для вычисления площадей: прямоугольного треугольника и прямоугольника.

2. Напишите программу, которая переводит переданное ей неотрицательное целое число в 10-значный восьмеричный код, сохранив лидирующие нули.

Вариант 8.

1. Найти все натуральные числа, не превосходящие заданного n, которые делятся на каждую из своих цифр.

2. Ввести одномерный массив A длиной m. Поменять в нём местами первый и последний элементы. Длину массива и его элементы ввести с клавиатуры. В программе описать процедуру для замены элементов массива. Вывести исходные и полученные массивы.

Вариант 9.

1. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т. д. Через сколько таких действий получится нуль?

2. Даны 3 различных массива целых чисел. В каждом массиве найти произведение элементов и среднеарифметическое значение.

Вариант 10.

1. На отрезке [100, N] (210 < N < 231) найти количество чисел, составленных из цифр a, b, c.

2. Составить программу, которая изменяет последовательность слов в строке на обратную.

Вариант 11.

1. Два простых числа называются «близнецами», если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары «близнецов» из отрезка [n, 2n], где n — заданное натуральное число, большее 2..

2. Даны две матрицы A и B. Написать программу, меняющую местами максимальные элементы этих матриц. Нахождение максимального элемента матрицы оформить в виде функции.

Вариант 12.

1. Два натуральных числа называются «дружественными», если каждое из них равно сумме всех делителей (кроме его самого) другого (например, числа 220 и 284). Найти все пары «дружественных» чисел, которые не больше данного числа N.

2. Даны длины сторон треугольника a, b, c. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Для вычисления медианы проведенной к стороне a, использовать формулу Вычисление медианы оформить в виде функции.

Вариант 13.

1. Натуральное число, в записи которого n цифр, называется числом Армстронга, если сумма его цифр, возведенная в степень n, равна самому числу. Найти все числа Армстронга от 1 до к.

2. Три точки заданы своими координатами X(x1, x2), Y(y1, y2) и Z(z1, z2). Найти и напечатать координаты точки, для которой угол между осью абсцисс и лучом, соединяющим начало координат с точкой, минимальный. Вычисления оформить в виде функции.

Вариант 14.

1. Составить программу для нахождения чисел из интервала [M, N], имеющих наибольшее количество делителей.

2.Четыре точки заданы своими координатами X(x1, x2), Y(y1, y2), Z(z1, z2), P(p1, p2). Выяснить, какие из них находятся на максимальном расстоянии друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде функции.

Вариант 15.

1. Найти все простые натуральные числа, не превосходящие n, двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.

2. Четыре точки заданы своими координатами X(x1, x2, x3), Y(y1, y2, y3), Z(z1, z2, z3), T(t1,t2, t3). Выяснить, какие из них находятся на минимальном расстоянии

друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде функции.

Вариант 16.

1. Составить программу для вычисления площади разных геометрических фигур.

2.Четыре точки заданы своими координатами X(x1, x2), Y(y1, y2), Z(z1, z2), P(p1, p2). Выяснить, какие из них находятся на максимальном расстоянии друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде функции.

Вариант 17.

1. Вычислить площадь правильного шестиугольника со стороной a, используя подпрограмму вычисления площади треугольника.

2. Пользователь вводит две стороны трех прямоугольников. Вывести их площади.

Вариант 18.

1. Даны катеты двух прямоугольных треугольников. Написать функцию вычисления длины гипотенузы этих треугольников. Сравнить и вывести какая из гипотенуз больше, а какая меньше.

2. Преобразовать строку так, чтобы буквы каждого слова в ней были отсортированы в обратном алфавитном порядке.

Вариант 19.

1. Даны длины сторон треугольника a, b, c. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Для вычисления медианы проведенной к стороне a, использовать формулу Вычисление медианы оформить в виде функции.

2. Задана окружность (x-a)2 + (y-b)2 = R2 и точки P(p1, p2), F(f1, f1), L(l1,l2). Выяснить и вывести на экран, сколько точек лежит внутри окружности. Проверку, лежит ли точка внутри окружности, оформить в виде функции.

Вариант 20.

1. Составить программу, которая изменяет последовательность слов в строке на обратную.

2. Напишите программу, которая выводит в одну строчку все делители переданного ей числа, разделяя их пробелами.

Вариант 21.

1. На отрезке [100, N] (210 < N < 231) найти количество чисел, составленных из цифр a, b, c.

2. Составить программу вычисления площади выпуклого четырехугольника, заданного длинами четырех сторон и диагонали.

Вариант 22.

1. Вычислить площадь правильного шестиугольника со стороной a, используя подпрограмму вычисления площади треугольника.

2. Напишите функцию, которая переводит переданное ей неотрицательное целое число в 10-значный восьмеричный код, сохранив лидирующие нули.

Вариант 23.

1. Найти все натуральные числа, не превосходящие заданного n, которые делятся на каждую из своих цифр.

2. Пользователь вводит две стороны трех прямоугольников. Вывести их площади.

Вариант 24.

1. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т. д. Через сколько таких действий получится нуль?

2. Найти все простые натуральные числа, не превосходящие n, двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.

Вариант 25.

1. Найти все простые натуральные числа, не превосходящие n, двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.

2. Составить программу, которая изменяет последовательность слов в строке на обратную.

Вариант 26.

1. Вычислить площадь правильного шестиугольника со стороной a, используя подпрограмму вычисления площади треугольника.

2. Пользователь вводит две стороны трех прямоугольников. Вывести их площади.

Вариант 27.

1. Найти все натуральные числа, не превосходящие заданного n, которые делятся на каждую из своих цифр.

2. Даны длины сторон треугольника a, b, c. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Для вычисления медианы проведенной к стороне a, использовать формулу Вычисление медианы оформить в виде функции.

Вариант 28.

1. Натуральное число, в записи которого n цифр, называется числом Армстронга, если сумма его цифр, возведенная в степень n, равна самому числу. Найти все числа Армстронга от 1 до k.

2. Составить программу, которая изменяет последовательность слов в строке на обратную.

Вариант 29.

1. Составить программу для нахождения чисел из интервала [M, N], имеющих наибольшее количество делителей.

2.Четыре точки заданы своими координатами X(x1, x2), Y(y1, y2), Z(z1, z2), P(p1, p2). Выяснить, какие из них находятся на максимальном расстоянии друг от друга и вывести на экран значение этого расстояния. Вычисление расстояния между двумя точками оформить в виде функции.

Вариант 30.

1. Найти все простые натуральные числа, не превосходящие n, двоичная запись которых представляет собой палиндром, т. е. читается одинаково слева направо и справа налево.

2. Составить программу для вычисления площади разных геометрических фигур.

Контрольные вопросы

1. Определение и вызов функций.

2. Синтаксис функций.

3. Использование параметров функций.

4. Использование оператора return.

5. Использование значения None.

6. Позиционные аргументы.

7. Аргументы — ключевые слова.

8. Значение параметра по умолчанию.

9. Получение аргументов — ключевых слов с помощью **.