КАЛУЖСКИЙ ФИЛИАЛ
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ Н.Э. БАУМАНА
(национальный исследовательский университет)»

Факультет «Информатика и управление»

Кафедра «Программное обеспечение ЭВМ, информационные технологии»

Высокоуровневое программирование

Лекция №8. «Повторное использование кода»

Калуга - 2020

Задачи для самостоятельного решения

1. С клавиатуры в одной строке вводится произвольное количество вещественных чисел. Запишите их в файл, расположив каждое число на отдельной строке.

2. Дан файл, полученный на выходе задачи №1:
   - загрузите список чисел;
   - вычислите их сумму и максимум и допишите их в файл.

   Выполнив программу несколько раз, убедитесь, что новые значения учитываются при подсчете.

Функции

- Функция - это блок кода, выполняющий определенные действия:
  - у функции есть имя, с помощью которого можно запускать этот блок кода сколько угодно раз
  - запуск кода функции называется вызовом функции
  - при создании функции, как правило, определяются параметры функции
  - параметры функции определяют, какие аргументы функция может принимать
  - функциям можно передавать аргументы
  - соответственно, код функции будет выполняться с учетом указанных аргументов

Сегментация данных

- **Статическая память** (выделяется до начала исполнения программы)
- **Динамическая память** (куча, heap – выделяется по запросу программиста)
- **Автоматическая память** (стековая - создание объектов автоматической памяти совершается компилятором, и компилятором эти же объекты разрушаются)

Механизм работы

- Большинство современных языков программирования для управления вызовом подпрограмм используют ***стек вызовов***.
- Примерный цикл работы стека вызова следующий:
  - Вызов подпрограммы создает запись в стеке; каждая запись может содержать информацию о данных вызова (аргументах, результате, а также адресе возврата).
  - Когда подпрограмма завершается, запись удаляется из стека и программа продолжает выполняться, начиная с адреса возврата.

Пример стека вызовов

```
def g():
    print('Inside g')

def f():
    print('Inside f')
    g()
    print('Inside f')

print('1. Inside module')
f()
print('2. Inside module')
```

```
1. Inside module
Inside f
Inside g
Inside f
2. Inside module
```

Пример получения стека через модуль *traceback* и его печати

```
32  import traceback
33
34  def f():
35      g()
36
37  def g():
38      for line in traceback.format_stack():
39          print(line.strip())
40
41  f()
```

```
File "C:/Projects/Py/SecondCourse/main.py", line 41, in <module>
    f()
File "C:/Projects/Py/SecondCourse/main.py", line 35, in f
    g()
File "C:/Projects/Py/SecondCourse/main.py", line 38, in g
    for line in traceback.format_stack():
```

Функции в Python

- функции создаются с помощью зарезервированного слова ***def***
- за ***def*** следуют имя функции и круглые скобки
- внутри скобок могут указываться параметры, которые функция принимает
- после круглых скобок идет двоеточие и с новой строки, с отступом, идет блок кода, который выполняет функция
- первой строкой, опционально, может быть комментарий, так называемая ***docstring***
- в функциях может использоваться оператор ***return***
- он используется для прекращения работы функции и выхода из неё
- чаще всего, оператор ***return*** возвращает какое-то значение

Преимущества и недостатки

Главное назначение подпрограмм сегодня - структуризация программы с целью удобства ее понимания и сопровождения.

Преимущества использования подпрограмм:

- декомпозиция сложной задачи на несколько более простых подзадач: это один из двух главных инструментов структурного программирование (второй - структуры данных);
- уменьшение дублирования кода и возможность повторного использования кода в нескольких программах - следование принципу DRY «не повторяйся» (англ. Don’t Repeat Yourself);
- распределение большой задачи между несколькими разработчиками или стадиями проекта;
- сокрытие деталей реализации от пользователей подпрограммы;
- улучшение отслеживания выполнения кода (большинство языков программирования предоставляет стек вызовов подпрограмм).

Недостатком использования подпрограмм можно считать накладные расходы на вызов подпрограммы, однако современные трансляторы стремятся оптимизировать данный процесс.

Виды функций

- Глобальные
  - Доступны из любой точки программного кода в том же модуле или из других модулей.
- Локальные (вложенные)
  - Объявляются внутри других функций и видны только внутри них: используются для создания вспомогательных функций, которые нигде больше не используются.
- Анонимные
  - Не имеют имени и объявляются в месте использования. В Python они представлены лямбда-выражениями.
- Методы
  - Функции, ассоциированные с каким-либо объектом (например, list.append(), где append() - метод объекта list).

Параметры и аргументы

Все параметры, указываемые в Python при объявлении и вызове функции делятся на:

- позиционные: указываются простым перечислением:

```
def function_name(a, b, c):  # a, b, c - 3 позиционных параметра
    pass
```

- ключевые: указываются перечислением ключ=значение:

```
def function_name(key=value, key2=value2):  # key, key2 - 2 позиционных аргумента
    pass                                    # value, value2 - их значения по умолчанию
```

- Позиционные и ключевые аргументы могут быть скомбинированы. Синтаксис объявления и вызова функции зависит от типа параметра, однако позиционные параметры (и соответствующие аргументы) всегда идут перед ключевыми

Объявление функции

```
def example_func(a, b, c):          # можно : 'a', 'b', 'c' - позиционные параметры
    pass

def example_func(a, b, c=3):        # можно : 'a', 'b' - позиционные параметры,
    pass                            #         'c' - ключевой параметр

def example_func(a=1, b=2, c=3):    # можно : 'a', 'b', 'c' - ключевые параметры
    pass

def example_func(a=1, c, b=2):      # нельзя: ключевой параметр 'a'
    pass                            #         идет раньше позиционнных
```

```python
def example_func(a, b, c=3):  # a, b - позиционные параметры, c - ключевой параметр
    pass

# Вызовы функции
example_func(1, 2, 5)          # можно : аргументы 1, 2, 5 распределяются
                               #         позиционно по параметрам 'a', 'b', 'c'

example_func(1, 2)             # можно : аргументы 1, 2 распределяются позиционно
                               #         по параметрам 'a', 'b'
                               #         в ключевой параметр 'c' аргумент
                               #         не передается, используется значение 3

example_func(a=1, b=2)         # можно : аналогично example_func(1, 2),
                               #         все аргументы передаются по ключу

example_func(b=2, a=1)         # можно : аналогично example_func(a=1, b=2),
                               #         если все позиционные параметры заполнены как
                               #         ключевые аргументы, можно не соблюдать порядок

example_func(c=5, b=2, a=1)    # можно : аналогично example_func(1, 2),
                               #         аргументы передаются по ключу

example_func(1)                # нельзя: для позиционного аргумента 'b'
                               #         не передается аргумент

example_func(b=1)              # нельзя: для позиционного аргумента 'a'
                               #         не передается аргумент
```

Преимущества ключевых параметров

- нет необходимости отслеживать порядок аргументов;
- у ключевых параметров есть значение по умолчанию, которое можно не передавать.

Упаковка и распаковка аргументов

В ряде случаев бывает полезно определить функцию, способную принимать любое число аргументов. Так, например, работает функция print(), которая может принимать на печать различное количество объектов и выводить их на экран.

Достичь такого поведения можно, используя механизм упаковки аргументов, указав при объявлении параметра в функции один из двух символов:

- *: все позиционные аргументы начиная с этой позиции и до конца будут собраны в кортеж;

- **: все ключевые аргументы начиная с этой позиции и до конца будут собраны в словарь.

Упаковка аргументов

```
def print_arguments(* some_tuple, ** some_dict):
    i = 1
    for item in some_tuple:
        print(f'{i} - {item}')
        i += 1


    for key, value in some_dict.items():
        print(f'{key}:  {value}')

print_arguments('Иванов', 'Петров', 'Сидоров', group='ITD-32', year=2020)
```

```
1 - Иванов
2 - Петров
3 - Сидоров
group:  ITD-32
year:  2020
```

Распаковка аргументов

Python также предусматривает и обратный механизм - распаковку аргументов, используя аналогичные обозначения перед аргументом:

- *: кортеж/список распаковывается как отдельные позиционные аргументы и передается в функцию;

- **: словарь распаковывается как набор ключевых аргументов и передается в функцию

Распаковка аргументов

```
def summ(*elements):
    result = 0
    for i in elements:
        result += i
    return result

print(f'sum = {summ(1, 2, 3, 4, 5)}')
print(f'sum = {summ(1, 2, 3, 4, 5, 10)}')
mas = [0, 2, 4, 6]
print(f'sum = {summ(*mas)}')
#такой вызов эквивалентен вызову summ(1, 2, 3, 4, 5)
```

```
sum = 15
sum = 25
sum = 12
```