Функции

Теория

Функции представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы.

Определение функции начинается с выражения def, которое состоит из имени функции, набора скобок с параметрами и двоеточия. Параметры в скобках необязательны. А со следующей строки идет блок инструкций, которые выполняет функция. Все инструкции функции имеют отступы от начала строки.

Пример 1

```
def имя_функции ([параметры]):
    инструкции

def Hello():
    print("Hello")

# Обратите внимание, что функция сначала определяется, а потом вызывается.
Hello()
```

Пример 2 Локальные функции

```
def print_messages():
    # определение локальных функций
    def One():
        print(name)
    def Two():
        print(family)
    # вызов локальных функций
    One()
    Two()

# Вызов функции print_messages
print_messages()
```

Пример 3 Передача параметров

```
# Если функция имеет несколько параметров, то необязательные параметры должны идти
после обязательных. Например:

def print_messages(name, family ="Corneplod"):
    # определение локальных функций
    def One():
```

```
        print(name)
    def Two():
        print(family)
    # вызов локальных функций
    One()
    Two()

# Вызов функции print_messages
print_messages("Victor")

# Результат вывода "Victor Corneplod"
```

Пример 4 Передача значений параметрам по имени. Именованные параметры

```
# несмотря на то, что параметр name идет первым в определении функции, мы можем
при вызове функции написать print_person(age = 22, name = "Tom") и таким образом
передать число 22 параметру age, а строку "Tom" параметру name.

def Test(name, age):
    print(f"Name: {name}  Age: {age}")


Test(age = 22, name = "Victor")

# Результат вывода "Name: Victor Age: 22"
```

Вариант 2

```
# Символ * позволяет установить, какие параметры будут именнованными - то есть
такие параметры, которым можно передать значения только по имени. Все параметры,
которые располагаются справа от символа *, получают значения только по имени:

# Если наоборот надо определить параметры, которым можно передавать значения
только по позиции, то есть позиционные параметры, то можно использовать символ /:
все параметры, которые идут до символа / , являются позиционными и могут получать
значения только по позиции

def testParam(name, age = 222, /, *, company):
    print(f"Name: {name}  Age: {age}  Company: {company}")


testParam("Victor", 16, company ="Google")   # Name: Victor  Age: 16  Company:
Google
testParam("Alex", 37, company ="MPT")        # Name: Alex  Age: 37  Company: MPT
testParam("Bin",company ="REY",)            # Bin  Age: 222  Company: REY
```

Пример 5 Неопределенное количество параметров

```
# С помощью звездочки можно определить параметр, через который можно передавать
неопределенное количество значений.

def Bruh(*numbers):
    print(numbers)

Bruh(23,42,25,2,1,5,14,1)
```

Списки 📃

Создание списка

```
# Оба этих определения списка аналогичны - они создают пустой список.

listOne = []
listTwo = list()

# Заполним его
listOne = [1, 41, 33]

# Для проверки элементов списка можно использовать стандартную функцию print,
которая выводит содержимое списка в удобочитаемом виде:
print(listOne)  # [1, 41, 33]

# Конструктор list может принимать набор значений, на основе которых создается
список:
hello = list("Hello")* 2
print(hello)      # ['H', 'e', 'l', 'l', 'o','H', 'e', 'l', 'l', 'o']

# Если необходимо создать список, в котором повторяется одно и то же значение
несколько раз, то можно использовать символ звездочки *, то есть фактически
применить операцию умножения к уже существующему списку:
hello = list("Hello")
print(hello)

# Обращение к элементам списка

listOne = [1, 41, 33]
print(listOne[0])
print(listOne[1])
print(listOne[2])
```

Методы и функции по работе со списками 🚗

```
item = "Max"
append(item): # добавляет элемент item в конец списка

insert(index, item): # добавляет элемент item в список по индексу index

extend(items): # добавляет набор элементов items в конец списка

remove(item): # удаляет элемент item. Удаляется только первое вхождение элемента.
Если элемент не найден, генерирует исключение ValueError

clear(): # удаление всех элементов из списка

index(item): # возвращает индекс элемента item. Если элемент не найден, генерирует
исключение ValueError

pop([index]): # удаляет и возвращает элемент по индексу index. Если индекс не
передан, то просто удаляет последний элемент.

count(item): # возвращает количество вхождений элемента item в список

sort([key]): # сортирует элементы. По умолчанию сортирует по возрастанию. Но с
помощью параметра key мы можем передать функцию сортировки.

reverse(): # расставляет все элементы в списке в обратном порядке

copy(): # копирует список

# Кроме того, Python предоставляет ряд встроенных функций для работы со списками:

len(list): # возвращает длину списка

sorted(list, [key]): # возвращает отсортированный список

min(list): # возвращает наименьший элемент списка

max(list): # возвращает наибольший элемент списка
```

Добавление и удаление элементов +

Для добавления элемента применяются методы append(), extend и insert, а для удаления - методы remove(), pop() и clear().

Использование методов:

```
# добавляем в конец списка
people.append("Alice")  # ["Tom", "Bob", "Alice"]
# добавляем на вторую позицию
people.insert(1, "Bill")  # ["Tom", "Bill", "Bob", "Alice"]
# добавляем набор элементов ["Mike", "Sam"]
```

```
people.extend(["Mike", "Sam"])      # ["Tom", "Bill", "Bob", "Alice", "Mike",
"Sam"]
# получаем индекс элемента
index_of_tom = people.index("Tom")
# удаляем по этому индексу
removed_item = people.pop(index_of_tom)     # ["Bill", "Bob", "Alice", "Mike",
"Sam"]
# удаляем последний элемент
last_item = people.pop()     # ["Bill", "Bob", "Alice", "Mike"]
# удаляем элемент "Alice"
people.remove("Alice")      # ["Bill", "Bob", "Mike"]
print(people)       # ["Bill", "Bob", "Mike"]
# удаляем все элементы
people.clear()
print(people)       # []
```

Перебор элементов массива

```
listOne = [1, 41, 33]
for i in listOne:
    print(i)

# Результат вывода
# 1
# 41
# 33
```

Задание 🏠

5

```
# Разделить операции +,-,/,* на функции передавая в них значения, посчитать в них
сумму и вернуть ответ (доработка первой практической с учетом ввода колличесва
чисел)
```

4

```
# Разделить операции +,-,/,* на функции передавая в них значения, посчитать в них
сумму и вернуть ответ (ввод двух чисел)
```

3

```
# Повторить все как в файле"
```