Функции

одна из основ программирования

может включать произвольное количество строк

именуется четко в соответствии с выполняемой задачей, английским глаголом. Например calculate, если она считает или add\*\*\* - если что-то добавляет

принцип – одна функция должна выполнять одно действие. Никаких «и» в описании ее сущности

функция начинается с ее объявления – define. Сокращенно def

*def* print\_hello(): # объявление функции

    ffff #тело – сколько угодно

    ddddd

    dfdsd

перед и после функции – пропуск по 2 строки

объявление не запускает функцию, для этого нужно к ней обратиться

print\_bye()

все функции принято писать в начале кода в алфавитном порядке, а затем уже к ним обращаться

функция может возвращать результат своего действия

return

Порядок работы интерпретатора:

- входит и считывает имена функций, не входя в них и записывает в name space

- потом начинает выполнять код с начала

- таким образом, если обращение к функции произошло до ее объявления в тексте кода не приведет к ошибке

Область видимости (scope resolution)

Функция – изолированная структура, т.о. переменные находящиеся в ней не имеют отношения к внешним и если обращаться к ним вне функции – то их имя будет - is not defined

Как только функция выполнится – то сборщик мусора почистит все переменные, которые были использованы внутри, оставив только return

- Локальная область видимости Local – внутри функции, недоступен извне

- Глобальная область видимости Global – код снаружи функции – доступен внутри функции

Чистая функция – на результат выполнения которой не имеется никакого внешнего влияния и ее результат не меняет ничего снаружи

- Между локальной и глобальной существует замкнутая Enclosed область видимости

например

*def* calculate\_sum():

    x = 2

    y = 3

*def* inner\_func():

        print(x)

        x = 10

        print(x)

    inner\_func()

result = calculate\_sum()

print(result)

должен вывести сначала х=2, затем х=10

    x = 2

    y = 3

являются замкнутая Enclosed для функции *def* inner\_func():

- самым высоким уровнем является built – in область видимости – это встроенные функции, выше чем глобальные переменные

Аргументы функции

В качестве аргументов могут служить любые типы данных (int, str, none) и функции

Функция, которая принимает в качестве аргумента другую функцию называется функцией высшего порядка

* Позиционные аргументы – передаваемые по их позиции в функции

def sum(x, y)

return x + y

result = sum(3, 2)

print(result)

* именованные аргументы или аргументы по ключу:

def diff(x, y)

return x - y

result = diff(y=3, x=2)

print(result)

или

def diff(x, y):

return x - y

result = diff(2, y =3)

print(result)

- Аргументы по умолчанию (которые присваиваются при объявлении функции):

def diff(x, y=5):

return x - y

result = diff(2)

print(result)

но при передаче аргумента, который был присвоен в начале – он имеет приоритет

def diff(x, y=5):

return x - y

result = diff(2, 10)

print(result)

- гибкое кол-во аргументов (flexible)

*def* calculate\_sum(\**args*):

    for arg in *args*:

        print(arg)

result = calculate\_sum(5, 66, 158)

print(result)

args – необязательное имя, главное \* и произвольное имя

- именованые аргументы - \*\*kwargs (keyword arguments) - тоже не обязательное, а рекомедованное имя

*def* calculate\_sum(\*\**kwargs*):

    return *kwargs*

result = calculate\_sum(*a*=5, *b*=10, *c*=8)

print(result)

{'a': 5, 'b': 10, 'c': 8}

*def* calculate\_sum(\**args*, \*\**kwargs*):

    return *args*, *kwargs*

result = calculate\_sum(1, "c", 5555, *a*=5, *b*=10, *c*=8)

print(result)

((1, 'c', 5555), {'a': 5, 'b': 10, 'c': 8})

рекурсия – разложение сложного на простые одинаковые элементы, до тех пор пока не будет найден необходимый объект

* doc string – документация, описание модуля, функции и т.д. записывается в начале

‘’’

My func

do nothing!!!!

‘’’

правила написания описаны в PEP