

PYTHON | БЕЗ ВОДЫ

Практический курс по Python

#Блок 5 | Функции

Для быстрого старта

О курсе

- ▶ Это ПРАКТИЧЕСКИЙ КУРС по программированию
- максимальная польза от курса в самостоятельном решении наибольшего числа задач
- вопросы на занятиях – самое ценное для учеников
- разбор задач учеников НА ЗАНЯТИЯХ
- разбор вопросов по задачам НА ЗАНЯТИЯХ

Регламент

- ▶ Длительность одного занятия 90 минут (может +)
- ▶ 1 занятие - теория | 3 занятия - практика
- ▶ Регулярность занятий 1 раз в неделю
- ▶ Проходим блоками
- ▶ Занятия по скайпу
- ▶ Запись занятий
- ▶ Видео доступно на youtube в закрытом плейлисте

#1 | Откуда потребность

Откуда потребность

- ▶ При разрастании программы
- ▶ При возникновении повторяющихся алгоритмов в коде программы
- ▶ При возникновении длинных проверок данных



Чем могут помочь функции

- ▶ Позволяют написать код раз и далее обращаться к нему в программе многократно

```
def square(x):  
    return x**2  
  
a = 8  
print(f'Квадрат восьми равен {square(a)}')  
  
b = 99  
print(f'Квадрат числа {a} равен {square(b)}')
```

Чем могут помочь функции

- ▶ В тестировании различных решений

```
from fnmatch import fnmatch
def sd(x):
    a = []
    for i in range(1,int(x**0.5)+1):
        if x%i==0:
            a.append(i)
            a.append(x//i)
    return sum(a)
k = 0
a = []
for x in range(10**7,99999,-1):
    if str(x)[0]=='9' and str(x)[-1]=='7':
        if fnmatch(str(x),'9?*55*7'):
            a.append((x,sd(x)%21))
            k+=1
    if k == 5:
        break
for x,y in sorted(a):
    print(x,y)
```

Чем могут помочь функции

- ▶ Сделать программу
 - понятнее
 - читабельнее
 - логичнее организованной
 - удобнее для проверок ошибок
(в т.ч. сторонними разработчиками)



Чем могут помочь функции

- ▶ В создании инструментов для текущего проекта
 - каждая функция решает отдельную задачу (одну)
 - функции могут писать отдельные разработчики
 - функции можно писать на будущее использование в проекте
 - функции можно брать из одних проектов и добавлять в другие

#2 | Что такое функция

Что такое функция?

► Функция:

- фрагмент программного кода, к которому можно обратиться из другого места программы
- С функцией связывается идентификатор, но многие языки допускают и безымянные функции
- По идентификатору функцию вызывают в программе, из одной программы в другой программе
- После выполнения функции управление возвращается обратно в адрес возврата — точку программы, где данная функция была вызвана, и программа пойдёт дальше.

Функции и ООП

▶ Процедурное программирование

- при написании программ используются функции

▶ Объектно – ориентированное программирование

- при написании программ используются объекты, методы и их свойства



#3 | Задание функции

Как задается функция?

- ▶ Конструкцией ***def название_функции(аргументы):***
 - Внутри функции аргументы называются ***параметрами***
- ▶ **Телом функции** - набором команд, каждая из которых отделена отступом от левого края на tab
- ▶ Конструкцией ***return***, которая возвращает определенное значение (по умолчанию *None*)

```
def simple(x):  
    for i in range(2,int(x**0.5)+1):  
        if x%i==0:  
            return False  
    return True
```

Как задается функция?

► Рекомендации по заданию функций

- Функции желательно называть максимально описательно
- Аргументы функции важно называть близко к их сути (словами)
- В имени функции используются буквы a-z, A-Z, цифры и символ нижнего подчеркивания
- Если нужно только наметить функцию, в теле функции пишут *pass*

```
def draw_triangle(height):  
    for i in range(1,height+1):  
        print('*'*i)  
draw_triangle(10)
```

```
block_5 x  
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3  
*  
**  
***  
****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

Как вызывается функция?

- ▶ Заданная в **def** функция сама по себе не вызывается
- ▶ Чтобы вызвать функцию, необходимо написать её название, после чего открыть и закрыть круглые скобки (с аргументом или без)

```
def multiply(a, b=10):  
    return a*b  
  
multiply(12) # 120  
multiply(2, 3) # 6
```

```
define('DRUPAL_ROOT', getcwd());  
  
require_once DRUPAL_ROOT . '/includes/bootstrap.inc';  
drupal_bootstrap(DRUPAL_BOOTSTRAP_FULL);  
menu_execute_active_handler();
```

- ▶ Функцию можно вызвать столько раз сколько нужно с новой строки без отступов

Виды функций по наличию параметров

▶ Функции без параметра

```
def f():  
    print('*****')  
    print('*      *')  
    print '**     **')  
    print('***    ***')  
    print('****   ****')  
    print('*****  *****')  
    print('*****  *****')  
    print('*****  *****')  
    print('***** *****')  
    print('***** *****')  
    print('***** *****')
```

▶ Функции с параметром (более гибкие)

```
def multiply(a, b=10):  
    return a*b  
  
multiply(12) # 120  
multiply(2, 3) # 6
```

Функции с параметрами

- ▶ Принимают на вход аргументы (arguments)
- ▶ Аргументы превращаются в переменные параметров (в теле функции)
- ▶ Количество аргументов функции определяет количество переменных, которые надо подать в функцию при вызове

```
def calculator(a,b,command):  
    if command == '+':  
        print(a + b)  
    if command == '-':  
        print(a - b)  
    if command == '*':  
        print(a * b)  
    if command == '/' and b != 0:  
        print(a / b)  
    if command == '/' and b == 0:  
        return 'На ноль делить нельзя!!!'
```

```
calculator(33,11)
```

block_4 ×

```
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3 /Users/raybin/Pycha  
Traceback (most recent call last):  
  File "/Users/raybin/PycharmProjects/course/block_4.py", line 35, in <module>  
    calculator(33,11)  
TypeError: calculator() missing 1 required positional argument: 'command'
```

Функции с параметрами

- ▶ Аргументы со значениями по умолчанию
 - При задании функции аргументам можно дать значения по умолчанию, что исключает задание им значений при вызове функции

```
def calculator(a,b,command="+"):  
    if command == '+':  
        print(a + b)  
    if command == '-':  
        print(a - b)  
    if command == '*':  
        print(a * b)  
    if command == '/' and b != 0:  
        print(a / b)  
    if command == '/' and b == 0:  
        return 'На ноль делить нельзя!!!'  
  
calculator(33,11)  
  
block_4 ×  
/Library/Frameworks/Python.framework/Versions/3.10/bin/python3  
44
```

*args и **kwargs

- ▶ *args используется для распаковки аргументов типа список, кортеж и т.п., позволяя вызывать функции со списком аргументов переменной длины

```
def printScores(student, *scores):  
    print(f"Student Name: {student}")  
    for score in scores:  
        print(score)  
  
printScores("Jonathan", 100, 95, 88, 92, 99)
```

25 ×

```
/Library/Frameworks/Python.framework/Versions  
Student Name: Jonathan  
100  
95  
88  
92  
99
```

*args и **kwargs

- ▶ **kwargs используется для распаковки аргументов типа словарь, позволяя вызывать функции со списком аргументов переменной длины

```
def printPetNames(owner, **pets):  
    print(f"Owner Name: {owner}")  
    for pet, name in pets.items():  
        print(f"{pet}: {name}")  
printPetNames("Jonathan", dog="Brock", fish=["Larry", "Curly", "Moe"], turtle="Sheldon")
```

Run

25 x

```
/Library/Frameworks/Python.framework/Versions/3.11/bin/python3 /Users/raybin/PycharmProjects/all/ege/25/25.py  
Owner Name: Jonathan  
dog: Brock  
fish: ['Larry', 'Curly', 'Moe']  
turtle: Sheldon
```

#4

Область видимости переменных

Области видимости переменных

▶ Локальные переменные

- Задаются внутри функции и **не видны** в основной программе

```
x = 7
def foo():
    x = 9
foo()
print(x)
```

block_5 ×

/Library/Frameworks/Python.framework

7

Области видимости переменных

▶ Глобальные переменные

- Задаются словом ***global*** внутри функции и **видны** в основной программе. Не рекомендуется их использовать, так как они затрудняют отладку и понимание программы.

```
x = 7
def foo():
    global x
    x = 9
foo()
print(x)
```

block_5 ×

/Library/Frameworks/Python.framework

9

Области видимости переменных

- ▶ Про списки и другие контейнеры для данных
- Задавать их словом *global* не надо - они **видны** в основной программе и в теле функции

```
a = []
def foo():
    a.extend([7,10,9,6,8])
foo()
print(a)
```

block_5 ×

/Library/Frameworks/Python.framework

[7, 10, 9, 6, 8]

#5 | Возвращаемые значения

Возвращаемые значения переменных

- ▶ Функция как переменная, что-то возвращает
 - Возвращаемое значение задается словом *return*
 - Функция может вернуть объект классов *int*, *float*, *str*, *list*, *tuple*, *bool* и прочих
 - Если в теле функции не написать *return* (что возможно), то она вернет *None*

```
print(print('Hello'))
```

```
block_5 ×
```

```
/Library/Frameworks/Python.framework
```

```
Hello
```

```
None
```

Возвращаемые значения переменных

▶ Функции для валидации данных

- Возвращают *bool* значение
- Называется максимально близко к проверяемому параметру
- Используется в основном коде программы для многочисленных проверок переменных (*is_even()*, *is_simple()*)
- В *return* записывается результат проверки

```
def is_simple(x):  
    return x > 1 and all(x%i for i in range(2, int(x**0.5) + 1))
```

Возвращаемые значения переменных

► ***return*** принудительно завершает функцию

- Аналог *break* в цикле
- Тело функции после *return* выполняться не будет
- Знание поведения *return* корректирует написание нами алгоритмов

```
def f(n):  
    if n == 50:  
        return 1  
    elif n > 50:  
        return 0  
    return f(n+2) + f(n*5)
```

Возвращаемые значения переменных

- ▶ Функция может возвращать несколько значений
 - После *return* можно указать список, словарь или кортеж
 - Можно указать возвращаемые значения через запятую – будет кортеж

```
def math(x,y):  
    return x + y, x - y, x * y, x // y  
  
print(math(33,8))
```

25 ×
/Library/Frameworks/Python.framework/Versions
(41, 25, 264, 4)

Виды функций по назначению

▶ Функции, возвращающие значения

- *sum(), sorted(), abs()*

▶ Функции, не предназначенные для возвращения результата

- *print()*

▶ Комбинированный вариант

- *input(), str(), list(), set()*

- мы сами выбираем, что функция должна выполнить и что вернуть

Рекомендации

- ▶ Не стоит создавать функции без особой надобности
 - негативно скажется на производительности программы
 - увеличивается количество строк кода
 - можно запутаться в своей библиотеке
- ▶ Не стоит решать несколько задач в одной функции
 - лучше писать на разные задачи разные функции
 - вызывать в одной функции другую (другие)

Что выделять в функцию

- ▶ Повторяющиеся последовательности команд
 - Полезно разбить длинную программу на составные части, с каждой из которых связать функцию
 - Тестирующие алгоритмы
 - Части кода, которые можно перенести в другие программы



Задача блока 5

▶ НАУЧИТЬСЯ ГРАМОТНО ПИСАТЬ ФУНКЦИИ

```
def a(b):  
    return a + b
```

```
def a(def b):  
    def(def(a))
```

```
def a(b(c)):  
    return b**c
```

```
def d(x):  
    def + x
```

