# Python

## Занятие 28. Конспект

### ПРОЦЕДУРЫ И ФУНКЦИИ

# Подпрограммы

**Подпрограмма** – это отдельная функционально независимая часть программы, имеющая имя и решающая отдельную задачу.

Подпрограммы:

- ✓ избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;
- ✓ улучшают структуру программы, облегчая ее понимание;
- ✓ позволяют распределять большие задачи между несколькими разработчиками или стадиями проекта;
- ✓ повышают устойчивость к ошибкам программирования и непредвидимым последствиям при модификациях программы.

# Два типа подпрограмм


В Python нет формального разделения подпрограмм на функции и процедуры (как, например, в Паскале или Си), и процедурой можно считать функцию, возвращающую пустое значение – в основном используется единственный термин – функция.

# Общий вид процедуры

**def**ine – определить

имя процедуры, даётся тем, кто пишет программу

```
def name_procedury(параметры):
    тело процедуры
```

**Параметр** – это переменная, от значения которой зависит работа подпрограммы.

если параметры не нужны ставят пустые скобки ()

пишут в скобках имена одной или нескольких переменных через запятую: (n) или (a,b,c,d)

# Вызов процедуры

При вызове процедуры нужно в скобках передать ей фактические значения параметров.

**Аргумент** – это значение параметра, которое передаётся подпрограмме при её вызове.

Аргументом может быть не только постоянное значение (число, символ), но также переменная, и даже арифметическое выражение.

```
name_procedury()
```

или

```
name_procedury(n)
```

или

```
name_procedury(a,b,c,d)
```

# Простая процедура

```
1  def printLine():
2      print("---------------------------------")
3  printLine()
4  print('эта процедура отделяет текст линиями')
5  printLine()
6  printLine()
```

процедура

основная часть программы

вызов процедуры

Оболочка

```
---------------------------------
эта процедура отделяет текст линиями
---------------------------------
---------------------------------
```

- ✓ можно вызывать сколько угодно раз
- ✓ нет дублирования кода
- ✓ изменять в одном месте

# Процедура с параметрами

```
1  def printLine(n):
2        print("=) "*n)
3  n=int(input('n= '))
4  printLine(n)
5  print('эта процедура рисует смайлики')
6  printLine(n)
```

Оболочка

```
n= 7
=) =) =) =) =) =) =)
эта процедура рисует смайлики
=) =) =) =) =) =) =)
```

символьная строка

```
1  def printLine(a,n):
2        print(a*n)
3  a=str(input('a= '))
4  n=int(input('n= '))
5  printLine(a,n)
6  print('эта процедура выводит символы')
7  printLine(a,n)
```

Оболочка

```
a= (*)
n= 10
(*)(*)(*)(*)(*)(*)(*)(*)(*)(*)
эта процедура выводит символы
(*)(*)(*)(*)(*)(*)(*)(*)(*)(*)
```

# Общий вид функции

define – определить

имя функции, даётся тем, кто пишет программу

```
def name_function(параметры):
    тело функции
    return (результат)
```

Выражение, стоящее после ключевого слова return будет возвращаться в качестве результата вызова функции.

Результат вызова функции можно:

- ✓ присвоить переменной,
- ✓ использовать его в качестве операндов математических выражений, т.е. составлять более сложные выражения.

**Функция** – это вспомогательный алгоритм, который возвращает результат (число, строку символов и др.).

# Функция

**Пример.** Написать функцию, которая вычисляет среднее арифметическое двух целых чисел.

исходные данные

```
def Avg(a, b):
    return (a+b)/2
a = 2; b = 5
print(Avg(a,b))
```

результат функции

**Ещё пример.**

```
1  def Avg(a, b):
2      return (a+b)/2
3  a = int(input('a= '))
4  b = int(input('b= '))
5  print(Avg(a,b))
6  if Avg(a,b) > 5:
7      print("Да!")
8  else:
9      print("Нет!")
```

Оболочка

```
a= 5
b= 4
4.5
Нет!
```

Что делает программа?

# Функция

```
1  def nod1 (a,b):
2      while a != 0 and b != 0:
3          if a > b:
4              a = a % b
5          else:
6              b = b % a
7      return a + b
8  print(nod1(int(input()),int(input())))
```

Оболочка ×

```
45
125
5
```

**Пример.** Написать функцию, которая вычисляет наименьший общий делитель двух целых чисел.

# Глобальные и локальные переменные

Переменные, которые введены в основной программе, называются **глобальными** (общими). Их **могут использовать все подпрограммы** (процедуры и функции).

Переменные, которые используются только внутри процедуры или функции, называются **локальными** (местными). К ним можно обращаться **только внутри этой подпрограммы, остальные** подпрограммы и основная программа **их не видят.** Такой прием называется **инкапсуляцией** (от латинского «помещение в капсулу»).

Локальная переменная создается только при вызове процедуры или функции. Как только работа подпрограммы будет закончена, все локальные переменные будут удаляться из памяти.

Имена локальных переменных в каждой подпрограмме можно выбирать независимо от имён локальных переменных других подпрограмм.

# Глобальные и локальные переменные

| **def show():**<br>print( s ) | **def showLocal():**<br>s = 7<br>print( s) | **def showGlobal():**<br>**global** s<br>s = 7<br>print( s ) |
|---|---|---|
| Процедура выводит значение **s**. После запуска транслятор сначала ищет локальную переменную с таким именем – её нет. Потом он начинает искать глобальную переменную: если такая переменная есть, на экран выводится её значение, если нет – будет выдано сообщение об ошибке. | Даже если существует глобальная переменная **s**, в первой строчке этой процедуры будет создана новая локальная переменная **s**, и её значение (7) появится на экране. | Эта процедура работает с глобальной переменной **s**. Она присвоит ей новое значение 7 (это «увидят» все остальные подпрограммы) и выведет его на экран. |

Нужно стараться **писать подпрограммы, которые не обращаются к глобальным переменным.**

# Глобальные и локальные переменные

В данном коде переменная b в функции локальная или глобальная?

```
def f(b):
    b += 0
    print(b)
b = 5
f(b)
```

Не запуская код, ответьте на вопрос: что выведет на экран данная программа?

```
def f(a):
    global b
    b += 3
    print(a + b)
b = 2
f(b)
```

```
def f():
    global a
    b = 2
    a, b = b, a
    print(a, b, end = " ")
a = 1
b = 2
f()
print(a, b, end = " ")
```

# Глобальные и локальные переменные

Не запуская код, выберите, какие из программ во время запуска получат ошибку выполнения.

```
def f():
    a = a
    print(a)
a = 5
f()
```

```
def f():
    print(a)
    a = a
a = 5
f()
```

```
def f(a):
    a = a
    print(a)
a = 5
f(a)
```

```
def f():
    print(a)
    global a
a = 5
f()
```

# Глобальные и локальные переменные

Не запуская код, ответьте на вопрос: что выведет на экран данная программа?

```
def f():
    global a
    global b
    b, c = a, b
def g():
        global a
        global d
        c = '0'
        a = d + c
```

```
a = '2'
b = '3'
c = '5'
d = '7'
f()
g()
f()
print(a + b + c + d)
```

# Функция

```
def binary(n):
    s = ''
    while n > 0:
        s = str(n%2) + s
        n //= 2
    return s

while 1:
    n = int(input())
    if n != 0:
        print(binary(n))
    else:
        break
```

Оболочка

```
7
111
5
101
6
110
```

**Пример.** Функция перевода десятичного числа в двоичное.

В основной ветке программы будем выполнять бесконечный цикл, в котором

1. запрашивается десятичное число,

2. если оно не ноль, то вызывается функция перевода его в двоичное представление и выводится результат работы функции на экран,

3. иначе (когда введен 0) будем прерывать цикл оператором break.

# Функция

```
def triangle(a, b, c):
    return a + b > c and a + c > b and b + c > a

a = int(input())
b = int(input())
c = int(input())
d = int(input())
if triangle(a,b,c) or triangle(a,b,d) \
  or triangle(a,c,d) or triangle(b,c,d):
    print("YES")
else:
    print("NO")
```

Оболочка

```
5
4
3
6
YES
```

**Пример.** Вам даны 4 отрезка. Выведите YES, если среди них найдутся 3, из которых можно составить треугольник, и NO в противном случае.

Для решения напишите функцию triangle(a, b, c), которая будет возвращать True, если из трёх заданных отрезков можно составить треугольник, и False иначе.

# Задачи

**1)** Напишите процедуру, которая принимает параметр – натуральное число N – и выводит на экран две линии из N символов "–".

**2)** Напишите процедуру, которая принимает один параметр – натуральное число *N*, – и выводит на экран прямоугольник длиной *N* и высотой 3 символа.

**3)** Напишите процедуру, которая выводит на экран треугольник со стороной N символов. При запуске программы N нужно ввести с клавиатуры.

**Пример:**

```
Длина цепочки: 7
-------
-------
```

**Пример:**

```
Длина прямоугольника: 7
ooooooo
o     o
ooooooo
```

**Пример:**

```
Сторона: 5
o
oo
ooo
oooo
ooooo
```

# Рекурсия

**Рекурсия** – это определение объекта через такой же объект (или объекты), но с другими параметрами.

**Рекурсивная подпрограмма** вызывает саму себя напрямую или через другие подпрограммы.

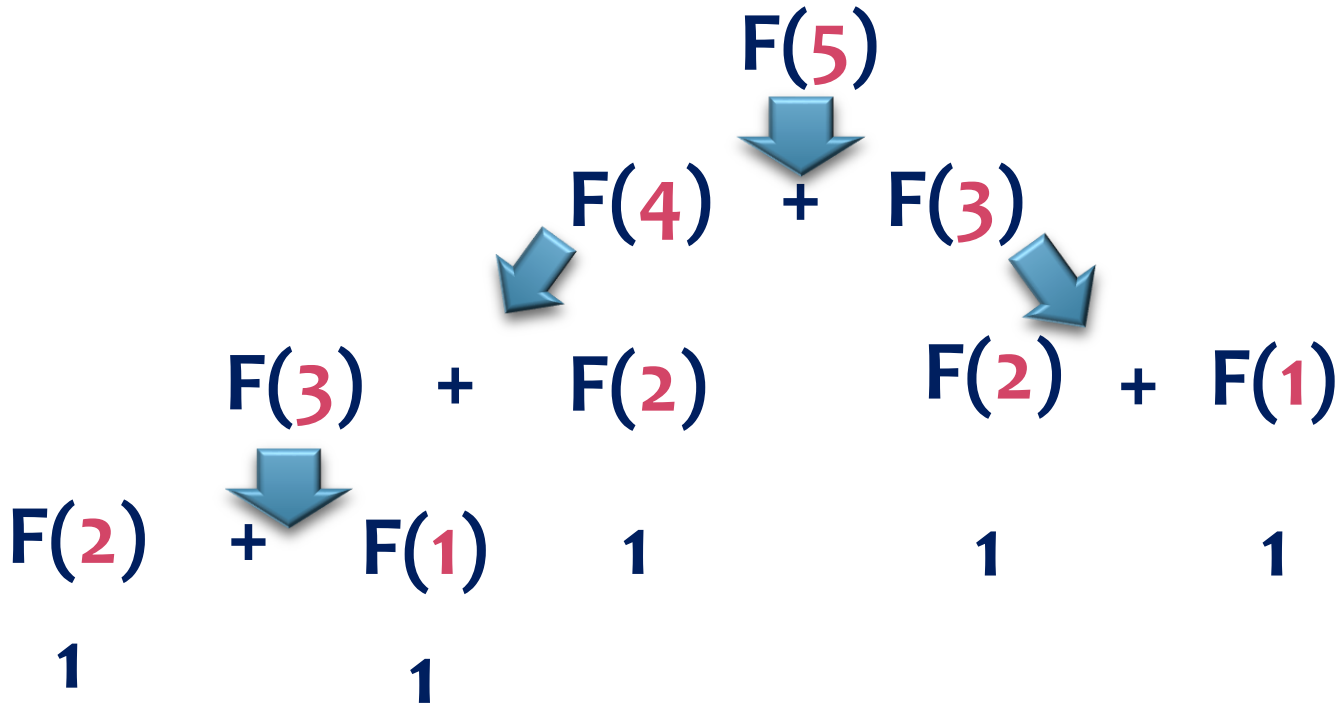
Количество вложенных вызовов функции или процедуры называется **глубиной рекурсии**. По умолчанию глубина рекурсии в языке Питон ограничена 1000 вызовов.

Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причем без явных повторений частей программы и использования циклов.

# Рекурсия

Рассмотрим рекурсию на примере:

```
def F(n):
        if n > 2:
                return F(n-1)+ F(n-2)
        else: return 1
n=int(input())
print(F(n))
```



# Примеры

Вычисление суммы натуральных чисел от 1 до n.

```
def sum(n):
    if n == 1:
        return 1
    else:
        return n + sum(n-1)
```

Быстрое возведение в степень

```
def power(a, n):
    if n == 0:
        return 1
    elif n % 2 == 1:
        return power(a, n - 1) * a
    else:
        return power(a, n // 2) ** 2
```

# Задачи

Чему равна сумма всех чисел, напечатанных на экране при выполнении вызова F(1)?

```
def F(n):
    print(n)
    if n < 5:
        F(n + 1)
        F(n + 3)
```

Чему будет равно значение, вычисленное при выполнении вызова F(6)?

```
def F(n):
    print(n)
    if n > 1:
        F(n - 1)
        F(n - 3)
```

```
def F(n):
  if n > 2:
    return F(n-1) + G(n-2)
  else: return n
def G(n):
  if n > 2:
    return G(n-1) + F(n-2)
  else: return n+1
```