КУРС «ИНФОРМАТИКА»

Часть 1. Рекурсия

2021 – 2022 УЧЕБНЫЙ ГОД

Рекурсия

Рекурсия – есть метод определения множества объектов или процесса в терминах самого себя.

Любое рекурсивное определение содержит две части:
- базисную часть
- Рекурсивную часть.

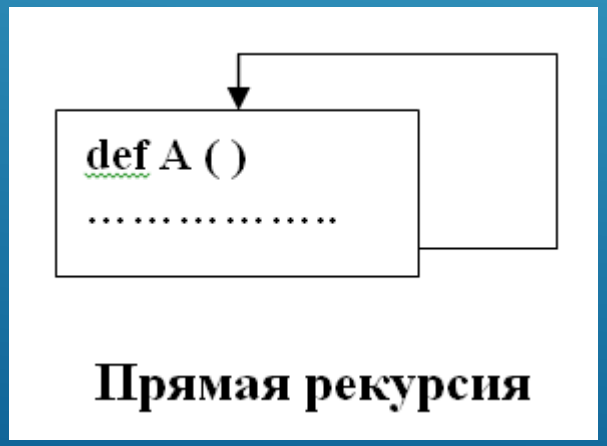
Базисная часть (условие завершения (одно или несколько)) является **нерекурсивным** утверждением, которое может быть вычислено для определенных параметров. Таким образом, базисная часть может задавать один или более случаев остановки рекурсии.

Рекурсивная часть определения записывается таким образом, чтобы при цепочке повторных применений утверждение из рекурсивной части приводилось бы к базисной части.

Виды рекурсивных функций

Рекурсивные алгоритмы реализуются через ***рекурсивные функции***

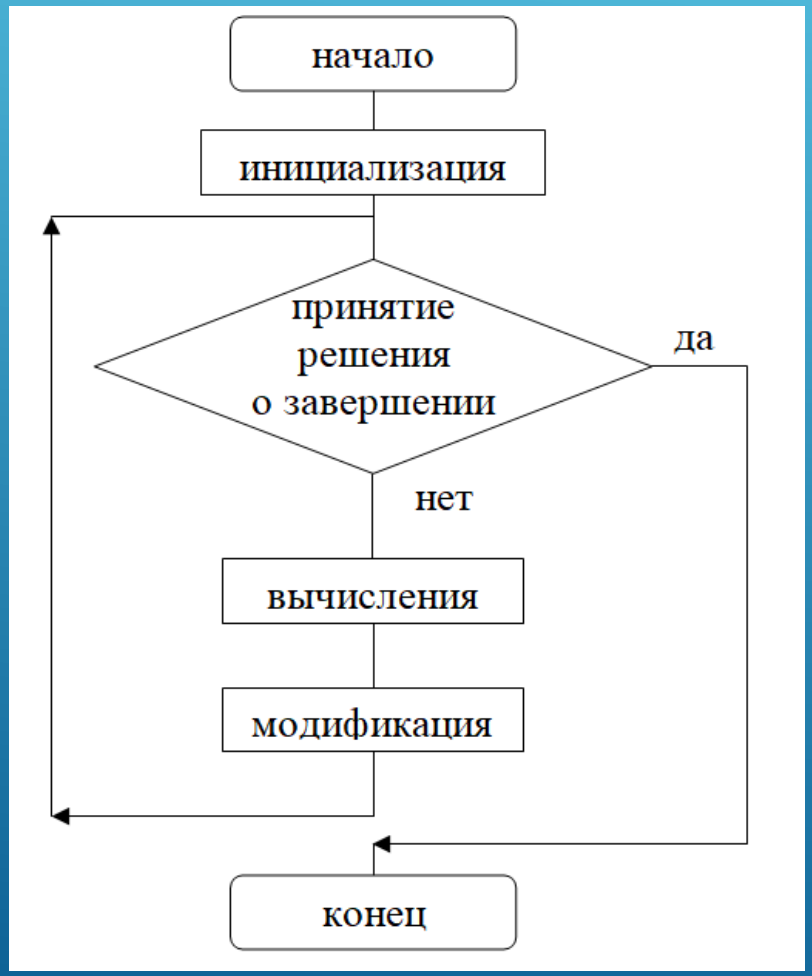
Прямая (явная) рекурсия характеризуется наличием в теле процедуры оператора обращения к ней же самой.
В случае ***косвенной (неявной) рекурсии*** одна процедура обращается к другой, которая (возможно через цепочку вызовов других процедур) вновь обращается к первой .
Далее будем рассматривать только прямую рекурсию.



Прямая рекурсия

Косвенная рекурсия

Итеративная схема организации вычислительного процесса



В основе итеративного вычислительного процесса лежат ***итеративные циклы*** *For и While*

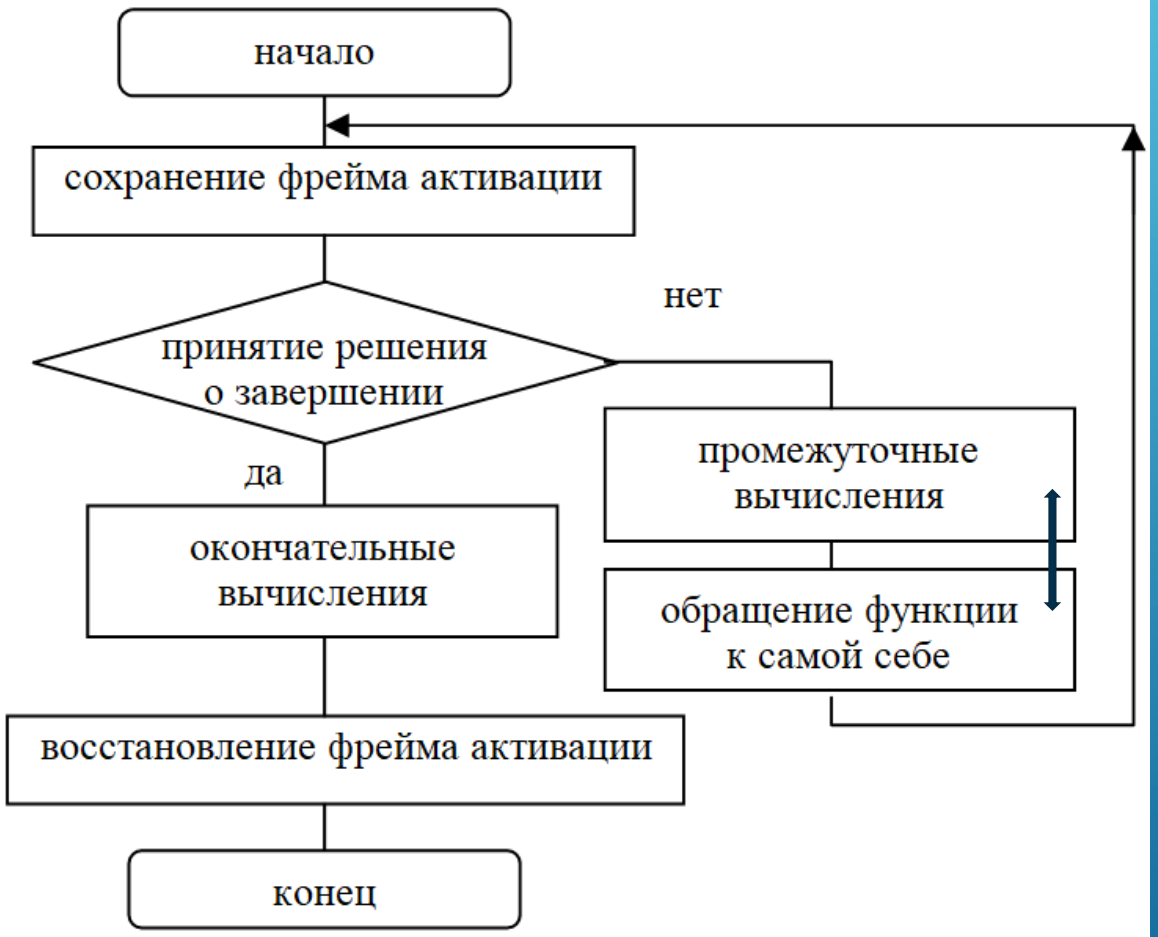
```
# Вычисление факториала n! = 1·2·3· ... ·n

def iterFact (n):
    i = 1                   # инициализация
    f = 1                   # инициализация
    while i <= n:           # решение о завершении
        f = f * i           # вычисления
        i +=1               # модификация
    return f

n = int(input())
print(n,"! = ", iterFact (n))
```

Соотношение между итерацией и рекурсией

Существует два важных положения, известных в математике и в программировании, определяющих соотношение между итерацией и рекурсией:

1. Любой итеративный цикл может быть заменен рекурсией.
2. Рекурсия не всегда может быть заменена итерацией.

Рекурсивная схема организации вычислительного процесса



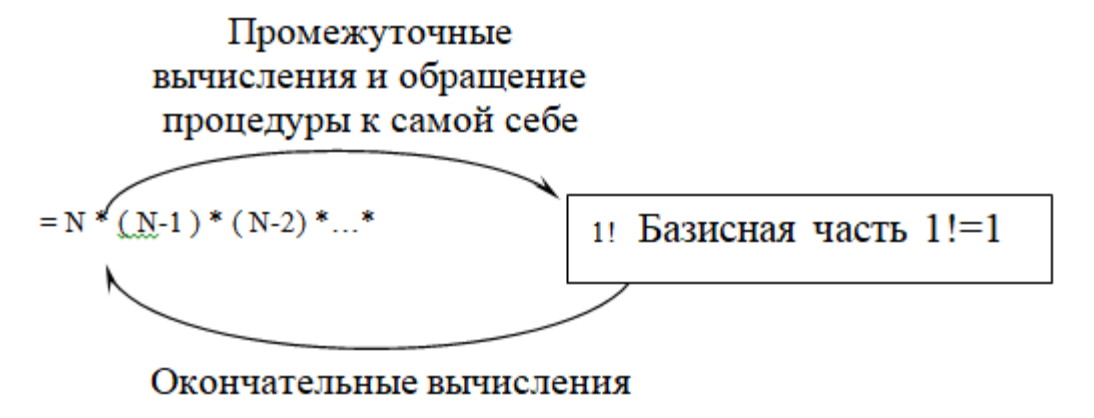

```
# Вычисление факториала n! = 1·2·3· ... ·n

def recFact (n):
  if n <= 1: #принятие решения о завершении  вычислений
    return 1 #окончательные вычисления для базисной части
  else:
    return recFact(n-1){1}*n # обращение функции к себе
                             # промежуточные вычисления

    #{2} - завершение активации

n = int(input())
print(n,"! = ", recFact (n))
```

{ 1 } – адрес возврата после завершения активации
{ 2 } – завершение активации.

Вычисление факториала

Рекурсивная схема вычисления факториала:

Базисная часть:
0! = 1; 1! = 1;
Рекурсивная часть:
N! = N * (N-1)! = N * (N-1) * (N-2)! = N * (N-1) * … * (N – (N-1))! =
= N * (N-1) * (N-2) *…*



Понятие глубины рекурсии

С каждым обращением к рекурсивной процедуре ассоциируется ***номер уровня рекурсии*** (номер фрейма активации). Считается, что при первоначальном вызове рекурсивной процедуры из основной программы номер уровня рекурсии равен единице.
Каждый следующий вход в процедуру имеет номер уровня на единицу больше, чем номер уровня процедуры, из которой производится это обращение.

Глубина рекурсии – максимальный уровень рекурсии в процессе вычисления при заданных аргументах.

В общем случае эта величина неочевидна, исключение составляют простые рекурсивные функции: например, при вычислении значения N! глубина рекурсии равна N.

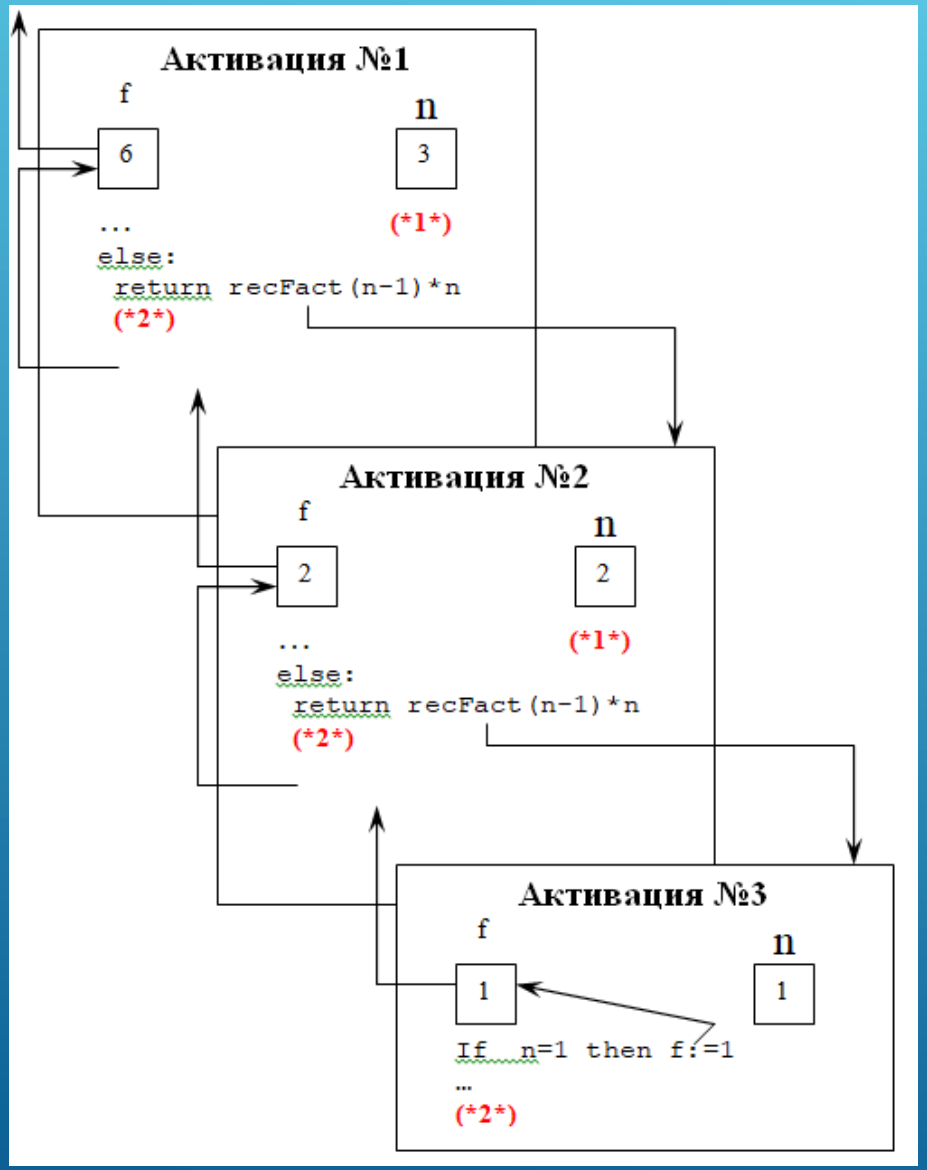
Фрейм активации

Так как обращаться к рекурсивной процедуре можно как из нее самой, так и извне, каждое обращение к рекурсивной процедуре вызывает ее **независимую активацию**.
При каждой активации **образуются копии всех локальных переменных и формальных параметров рекурсивной процедуры**, в которых “оставляют следы” операторы текущей активации.
Таким образом, для рекурсивной процедуры может одновременно существовать несколько активаций.
Для обеспечения правильного функционирования рекурсивной процедуры **необходимо сохранять адреса возврата** в таком порядке, чтобы возврат после завершения каждой текущей активации выполнялся в точку, соответствующую оператору, непосредственно следующему за оператором рекурсивного вызова.

- Совокупность локальных переменных, значений фактических параметров, подставляемых на место формальных параметров рекурсивной процедуры, и адреса возврата однозначно характеризует текущую активацию и образует ***фрейм активации***.

Фрейм активации необходимо сохранять при очередной активации и восстанавливать после завершения текущей активации. Так как при выходе из текущей активации самым первым должен быть восстановлен фрейм, который был позже всех сохранен, для хранения фреймов используется автоматическая память, т.е. область системного стека.

Фреймы активации при вычислении 3!




```
# Вычисление факториала n! = 1·2·3· ... ·n

def recFact (n):
  if n <= 1: #принятие решения о завершении  вычислений
    return 1 #окончательные вычисления для базисной части
  else:
    return recFact(n-1){1}*n # обращение функции к себе
                             # промежуточные вычисления
    #{2} - завершение активации

n = int(input())
print(n,"! = ", recFact (n))
```

Примеры рекурсивных функций

С помощью рекурсивной функции определить, является ли число четным или нечетным

```
def f(x):
    if x==0:
        print ("четное")
    elif x==1:
        print ("нечетное")
    elif x>0:
        f(x-2)
    elif x < 0:
        f(x+2)
n = int(input())
f(n)
```

```
2022
четное
```

```
2023
нечетное
```

```
2024
Traceback (most recent call last):
 RecursionError: maximum recursion depth exceeded while pickling an object
```

```
-2022
четное
```

```
-2023
Traceback (most recent call last):
  RecursionError: maximum recursion depth exceeded while pickling an object
```

Примеры рекурсивных функций

С помощью рекурсивной функции вычислить n-степень положительного числа x

```
def f(x, n):
    if n==0:
        return 1
    elif n > 0:
        return f(x, n-1)*x
    elif n < 0:
        return f(x, n+1)/x

print("Введите число x: ")
x = int(input())
print("Введите число n:")
n = int(input())
print("Результат =",f(x, n))f(n)
```

```
Введите число x:
2
Введите число n:
5
Результат = 32
```

```
Введите число x:
10
Введите число n:
-2
Результат =  0.01
```

Примеры рекурсивных функций

С помощью рекурсивной функции вычислить n-степень положительного числа x

```
def f(x, n):
    if n==0:
        return 1
    elif n > 0:
        return f(x,n-1)*x
    elif n < 0:
        return f(x,n+1)/x

print("Введите число x: ")
x = int(input())
print("Введите число n:")
n = int(input())
print("Результат =",f(x, n))f(n)
```

```
Введите число x:
2
Введите число n:
5
Результат = 32
```

```
Введите число x:
10
Введите число n:
-2
Результат =  0.01
```

Пример рекурсивных функций

Исполнитель преобразует число на экране. У исполнителя есть две команды, которым присвоены номера:

1. Прибавить 1

2. Умножить на 2

Первая команда увеличивает число на экране на 1, вторая умножает его на 2. Программа для исполнителя – это последовательность команд. Сколько существует программ, для которых при исходном числе 1 результатом является число 20, и при этом траектория вычислений содержит
число 10?

```
def f(x, y ):
    if x == y: return 1
    elif x > y : return 0
    elif if x < y:
     return f(x+1,y)+f(x*2,y)

print(f(1,10)*f(10,20))
```