Основы и методология программирования

Семинар 5

Функции
Рекурсия

Функции и структурное программирование

Структурное программирование — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков.

1. Любая программа представляет собой структуру, построенную из трёх типов базовых конструкций: **последовательное** исполнение; **ветвление**; **цикл.**

2. **Метод нисходящего проектирования** предполагает последовательное разложение общей функции обработки данных на простые функциональные элементы («сверху-вниз»). В результате строится иерархическая схема, отражающая состав и взаимоподчиненность отдельных функций.

3. **Модульное программирование** является естественным следствием проектирования сверху вниз и заключается в том, что программа разбивается на части – модули, разрабатываемые по отдельности. Модуль – это самостоятельная часть программы, имеющая определенное назначение и обеспечивающая заданные функции обработки автономно от других программных модулей.

Функции и структурное программирование

4. Модуль должен обладать следующими свойствами:

- **один вход и один выход** – на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатных данных;
- **функциональная завершенность** – модуль выполняет перечень регламентированных операций для реализации каждой отдельной функции в полном составе, достаточных для завершения начатой обработки;
- **логическая независимость** – результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей;
- **слабые информационные связи с другими программными модулями** – обмен информацией между модулями должен быть по возможности минимизирован; обозримый по размеру и сложности программный код.

Функции

```
def myFunc(str1):   # str1 - обязательный параметр при вызове функции
    """             # описание функции (опционально)
    Эта функция делает много полезного
    :param str1: один входной параметр
    :return: возвращает строку наоборот
    """
    str2 = ""
    i = 1
    while (len(str2) < len(str1)):
        str2 += str1[-i]
        i += 1
    str1 = "AAA"
    print("id str1 =", id(str1))
    return str2  # возвращаемое значение (опционально)
```

Функции

```
def get_person_info(name, age):     # Порядок вывоза важен, get_person_info(23, "Ivan")-
некорректно
   print (name, "is", age, "years old")
# Хотя в описании функции первым аргументом идет имя, мы можем вызвать функцию вот так
get_person_info(age=23, name="John")
```

```
def empty(agr1 = 30):   #аргументы могут задаваться параметрами по умолчанию
                        # print(agr1)
   pass                 # можно использовать когда функция не готова, но требуется ее объявить
print(empty())          # None, т.к. функция ничего не возвращает
```

```
func = lambda x, y: x + y  # функции могут быть анонимными
func(1, 2)                 # при этом они остаются функциями
func('a', 'b')             # не строгая типизация, это плюс и минус
#func (1, 'a')             # а вот это уже ошибка! Нельзя складывать int и string
(lambda x, y: x + y)(1, 2) # можно объединять объявление и вызов анонимной функции
```

Задача 1

Написать функцию для расчета сложных процентов.
Параметры: начальный капитал, годовая процентная ставка, число лет (вклада).
Возвращаемое значение – итоговая сумма в конце срока вклада.
В основной программе ввести начальный капитал (больший нуля), процентную ставку и число лет. Вывести таблицу значений итоговых сумм в конце каждого года до заданного числа лет.

Задача 1

```
def countPercent(start, percent, n):
    sum = 0.0
    term = 1
    i = 1
    while i <= n:
        term *= 1 + percent / 100
        sum = start * term
        i += 1
    return sum

startMoney = 100.0  # float(input("Сколько было? "))
percent = 10.0  # float(input("Под какой процент? "))
years = 10  # int(input("На сколько полных лет? "))
i = 1
while (i <= years):
    print("Выплаты за %ій год %g" % (i, countPercent(startMoney, percent, i)))
    i += 1
```

Задача 2

Определить, как часто встречается определенный символ в строке. Вывести абсолютное и относительное значение. Подсчет оформить в виде функции.

```
def countAbs(str1, sym):
    i = 0
    count = 0
    while i < len(str1):
        if (str1[i] == sym):
            count += 1
        i += 1
    return count

strInput = input("Введите строку: ")
i = 0
while i < len(strInput):
        symb = strInput[i]
        absFreq = countAbs(strInput, symb)
        print("Symbol %c, abs_count = %i %i rel_count=%g"
% (symb, absFreq, absFreq1, absFreq / len(strInput)))
        i += 1
```

Рекурсия

Рекурсией называется процесс вызова функцией самой себя.
Типичным примером рекурсии может послужить функция вычисления факториала числа.

$$N! = (N-1)! * N$$
$$1! = 1$$

```
def fact(n):
    return 1 if n == 1 else fact(n-1) * n

print("10! =", fact(10)) # 10! = 3628800
```

Другой пример - математические последовательности (или ряды), где следующий член зависит от предыдущего или предыдущих. Например, геометрическая прогрессия.

$$b_{n+1} = b_n * q$$

```
b1 = 1
def geomProgr(n, q):
    return n if n == b1 else geom_progr(n-1, q) * q

print("B5 =", geom_progr(5, 2))   # B5 = 16
```

Задача 3

Задана функция:

$$A(m,\ n) = \begin{cases} n+1, & m = 0; \\ A(m-1,\ 1), & m > 0,\ n = 0; \\ A(m-1,\ A(m,\ n-1)), & m > 0,\ n > 0. \end{cases}$$

Даны два целых неотрицательных числа m и n, каждое в отдельной строке. Выведите A(m,n).

Задача 3

```
def recursiveFunc(m, n):
    if m == 0: # базовый случай
        return n + 1
    elif n == 0 and m > 0:
        return recursiveFunc(m - 1, 1)
    else:
        return recursiveFunc(m - 1, recursiveFunc(m, n - 1))

m = int(input("Введите m: "))
n = int(input("Введите n: "))
print("Значение функции A(%d, %d)=%d" % (m, n, recursiveFunc(m, n)))
```

Задача 4 “Алфавит”

Вводятся 2 строки : например, “aaa” и “abd” одинаковой длины. Нужно вывести полный перебор строк от первой до второй в алфавитном порядке, т.е.

aaa
aab
aac
...
aaz
aba
...
abd

Задача 4 “Алфавит”

```
def getNext(str1):
    i = 1
    retVal = ""
    if (str1[-1] != 'z'):
        return str1[:-i] + chr(ord(str1[-1]) + 1)

    while str1[-i] == 'z':
        i += 1
    if (i == len(str1) + 1): return ""  # если уже zzz

    retval = str1[:-i + 1]
    retval = getNext(retval)
    retval += 'a' * (i - 1)
    return retval

str1 = "aav"
str2 = "aba"

print(str1)
tmp = getNext(str1)
while tmp != str2:
    print(tmp)
    tmp = getNext(tmp)
print(str2)
```