ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Python. Оператор цикла с заголовком. Вычисление конечных сумм и произведений

Методические указания к лабораторной работе

6

Рязань 2017

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Python. Оператор цикла с заголовком. Вычисление конечных сумм и произведений

Лабораторная работа №6

Методические указания к лабораторной работе

Рязань 2017

УДК 004.432

Python. Оператор цикла с заголовком. Вычисление конечных сумм и произведений / Рязан. гос. радиотехн. универ.; Сост.: А.Н. Пылькин, Н.Н. Степанов, Н.А. Тярт. – Рязань, 2017 г. – 20 с.

Рассмотрены вопросы проектирования алгоритмов и программ циклической структуры с заголовком. Описаны основные приемы использования оператора цикла с заголовком (оператора for), а также применение функции range, операторов continue и break, конструкции else для реализации циклов.

Для задач вычисления конечных сумм и произведений предложено использовать три различных алгоритма в зависимости от вида слагаемых (сомножителей) конечной суммы (произведения).

Ил.: 6. Бибилиогр.: 4 назв.

Печатается по решению Научно-методического совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра информатики, информационных технологий и защиты информации ФГБОУ ВО «Липецкий государственный педагогический университет им. П.П. Семенова-Тян-Шанского» (зав. каф., к.т.н., доц. Скуднев Д.М.).

Цель работы

Получение практических навыков реализации цикла с заголовком с использованием операторов `for`, `continue` и `break`, функции `range` и конструкции `else`.

Оператор цикла с заголовком

Оператор цикла с заголовком (оператор `for`) в языке Python считается менее универсальным, чем оператор цикла с предусловием (оператор `while`). Достоинством цикла `for` является более высокое быстродействие при выполнении циклической программы. Оператор обладает способностью последовательно перебирать элементы любого упорядоченного типа, используемого в качестве параметра цикла.

В общем виде оператор цикла с заголовком можно представить следующим образом:

```
for <параметр> in <список значений>:
    <команда>
    ...
    <команда>
```

Например, в результате выполнения программы

```
for i in 1, 2, 3, 'один', 'два', 'три':
    print(i)
```

выводятся результаты

```
1
2
3
один
два
три
```

В процессе выполнения программы переменная `i` принимает значения различных типов (целочисленного и строкового типов). Этот факт демонстрирует следующий пример:

```
for i in 1, 2, 3, 'один', 'два', 'три':
    print(i*2)
```

Результат выполнения этого оператора цикла с заголовком:

```
2
4
6
одинодин
двадва
тритри
```

Параметром цикла `for` может быть перечисление, что заимствовано из других языков. В этом случае перечисление в операторе `for` можно рассматривать в качестве аналога оператора foreach, например, в языке C#. В обоих случаях оператор повторяет цикл для каждого элемента массива или коллекции объектов (более подробно определение этих терминов будет приведено в дальнейших разделах).

Например, если параметром цикла являются упорядоченные значения массива, то в результате выполнения программы

```
a = [1, 2, 3, 4, 5]
for i in a:
    print(i*2)
```

выведутся следующие значения:

```
2
4
6
8
10
```

Функция range

Функция range позволяет организовать повторение некоторой совокупности действий заданное число раз или указывает на изменение значения параметра цикла от некоторого начального значения до конечного значения. В первом случае заголовок цикла имеет следующий формат:

```
for i in range(n):
    <команда>
    ...
    <команда>
```

В данном случае цикл выполняется для значений параметра i = 0, 1, 2, …, n−1. В качестве n можно использовать целочисленную константу, переменную или целочисленное арифметическое выражение. Если значение n отрицательное или равно нулю, то команды, образующие тело цикла, не выполнятся ни разу.

Например, в результате выполнения операторов

```
n = 3
for i in range(n):
    print(i)
```

выведутся значения:

```
0
1
2
```

Индексация в функции range(n) начинается с нулевого значения, однако допускается изменение параметра цикла с любого значения. Например, оператор

```
for i in range(1, 3):
    print(i)
```

выводит значения:

```
1
2
```

В общем случае формат цикла с заголовком и использованием функции range имеет вид:

```
for i in range(<выражение 1>, <выражение 2>, <выражение 3>):
    <команда>
    ...
    <команда>
```

Здесь <выражение 1> задает начальное значение параметра цикла; <выражение 2> определяет значение параметра, следующего за конечным значением; а <выражение 3> – это шаг изменения параметра цикла. Существует возможность указать начальное значение больше конечного и отрицательный шаг, тогда значения будут перебираться в порядке убывания.

Если <выражение 1> >= <выражение 2> при положительном <выражение 3>, то цикл не выполнится ни разу.

Табулирование функции с помощью оператора цикла с заголовком

<u>Пример 1.</u> Ранее рассматривалась задача табулирования следующей функции $y = f(x)$:

$$f(x) = \begin{cases} 0, & x \le 0; \\ x, & 0 < x < 1; \\ 1, & x \ge 1, \end{cases}$$

при изменении аргумента x по закону $x = x_0(h_x)x_n$.

Поскольку в качестве параметра в операторе `for` не может быть использована переменная вещественного типа (например, переменная x), то введем дополнительную переменную i, значение которой будет изменяться от 1 до N_x с постоянным шагом 1. Значение N_x равно числу повторений цикла при законе изменения параметра $x = x_0(h_x)x_n$ и определяется формулой

$$N_x = \left[\frac{x_n - x_0}{h_x}\right] + 1,$$

где [ ] означает целую часть.

Для переменной x перед циклом зададим ее начальное значение x_0, а в теле цикла будем производить ее модификацию (изменение). Закон изменения параметра i цикла укажем в заголовке цикла. В результате получаем алгоритм циклической структуры с заголовком (рис. 1).

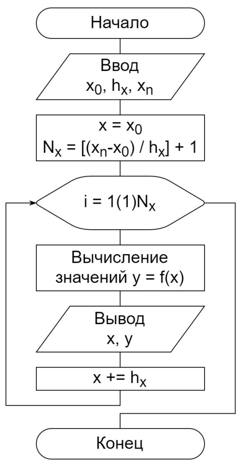

Рис. 1. Алгоритм табулирования функции

Программа, реализующая данный алгоритм, имеет вид:

```
from math import *

# Цель: табулирование функции y = F(x)
#   с помощью оператора цикла с параметром.
# Переменные:
#     x - переменная цикла;
#     x0, xn - начальное и конечное значения;
#     hx - шаг изменения;
#     i - параметр цикла;
#     nx - число повторений тела цикла.
# Программист: Степанов Н.Н.
# Дата написания: 25.02.2017.

print("Введите исходные данные:")
print("x0 = ", end='')
x0 = float(input())
print("hx = ", end='')
hx = float(input())
print("xn = ", end='')
xn = float(input())

print("Вы ввели:")
print("x0 = %.2f   hx = %.2f   xn = %.2f" % (x0, hx,
xn))

x = x0
nx = trunc((xn - x0) / hx + 1E-6) + 1
print("nx = ", nx)
print("Результат:")

for i in range(1, nx + 1):
    if x <= 0:
        y = 0
    elif x < 1:
        y = x
    else:
        y = 1
    print("X = %.3f   Y= %.3f" % (x, y))
    x += hx
```

В программе использована функция `trunc(x)` из модуля `math`, результат которой есть наибольшее целое число, меньшее или равное `x`. Аргумент функции `trunc` дополнен слагаемым 1E–6, которое, не изменяя полученного результата, позволяет избежать ошибки представления вещественных значений x_0, h_x и x_n.

Алгоритм вычисления конечных сумм и произведений

Конечная сумма представляет собой сумму некоторого числа членов функционального ряда и в общем случае имеет вид:

$$s = t_0(x) + t_1(x) + \cdots + t_m(x) = \sum_{n=0}^{m} t_n(x).$$

В процессе вычисления суммы реализуется алгоритм накопления слагаемых $t_n(x)$ в некоторой переменной s при изменении индекса $n = 0(1)m$. Этот процесс накопления целесообразно реализовывать с применением оператора цикла с параметром n, изменяющимся от 0 до m с целочисленным шагом 1.

В зависимости от вида слагаемого $t_n(x)$ можно выделить три основных алгоритма вычисления конечной суммы.

Алгоритм №1. В простейшем случае осуществляется непосредственное вычисление слагаемого $t_n(x)$, тогда алгоритм имеет вид, представленный на рис. 2, а.

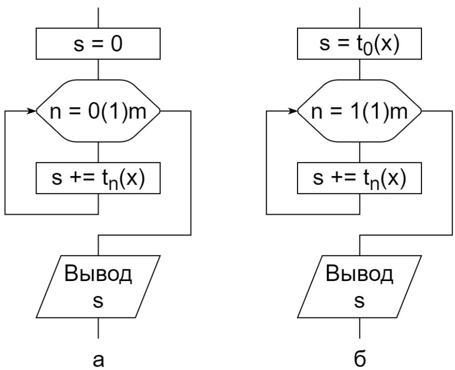

Рис. 2. Простейший алгоритм №1 вычисления конечной суммы

Одной из модификаций алгоритма №1 является схема, в которой первоначальное значение суммы S не равно нулю. Например, алгоритм, в котором перед циклом в переменную *S* занесено значение первого слагаемого $t_n(x)$ и цикл с параметром реализует закон изменения параметра $n = 1(1)m$, показан на рис. 2, б.

Алгоритм №2. Данный алгоритм применяется в тех случаях, когда «прямое» вычисление слагаемого приводит к достаточно громоздким вычислениям и связаны с оперированием большими значениями. В подобных случаях применим алгоритм, когда слагаемое $t_n(x)$ вычисляется с помощью рекуррентной формулы. Алгоритм вычисления конечной суммы такого типа показан на рис. 3.

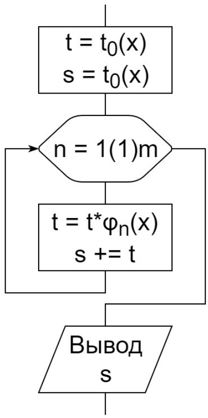

Рис. 3. Алгоритм №2 вычисления конечной суммы

Например, пусть необходимо вычислить сумму:

$$S = \sum_{n=0}^{m} (-1)^n \frac{x^{2n}}{(2n)!}.$$

Общий вид слагаемого в таком случае:

$$t_n(x) = (-1)^n \frac{x^{2n}}{(2n)!}.$$

Для вычисления суммы *s* по схеме алгоритма №2 находим:

$$t_0(x) = (-1)^0 \frac{x^{2*0}}{(2*0)!} = 1.$$

Вычисление текущего слагаемого реализуем с помощью следующей рекуррентной формулы:

$$t_n(x) = t_{n-1}(x)\varphi_n(x),$$

тогда сомножитель:

$$\varphi_n(x) = \frac{t_n(x)}{t_{n-1}(x)},$$

$$t_{n-1}(x) = (-1)^{n-1}\frac{x^{2(n-1)}}{[2(n-1)]!} = (-1)^{n-1}\frac{x^{2n-2}}{(2n-2)!}.$$

В результате находим:

$$\varphi_n(x) = \frac{t_n(x)}{t_{n-1}(x)} = \frac{(-1)^n \frac{x^{2n}}{(2n)!}}{(-1)^{n-1}\frac{x^{2n-2}}{(2n-2)!}} = \frac{(-1)^n x^{2n}(2n-2)!}{(-1)^{n-1}x^{2n-2}(2n)!} =$$

$$= \frac{(-1)x^2(2n-2)!}{(2n)!} = \frac{-x^2 * 1 * 1 * \dots * (2n-2)}{1 * 2 * \dots * (2n-2)(2n-1)(2n)} = \frac{-x^2}{(2n-1)(2n)}.$$

Алгоритм для рассматриваемого примера показан на рис. 4.

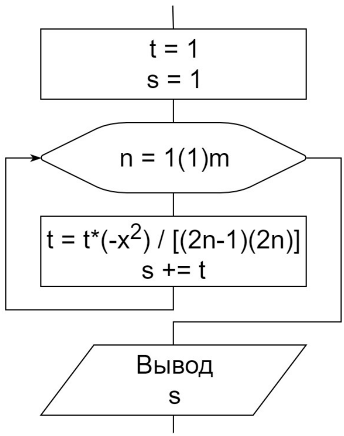

Рис. 4. Пример «рекуррентного» алгоритма

Алгоритм №3. Алгоритм этого типа является комбинированием алгоритмов №1 и №2 для тех случаев, когда слагаемое $t_n(x)$ можно представить в виде:

$$t_n(x) = A_n(x)B_n(x).$$

При этом один из сомножителей (например, $A_n(x)$) может быть определен напрямую, а другой сомножитель $B_n(x)$ требует использования рекуррентной формулы. Например, рассмотрим алгоритм вычисления конечной суммы:

$$S = \sum_{n=1}^{m}\frac{x^n}{n}.$$

В данном случае

$$t_n(x) = \frac{x^n}{n} = A_n(x)B_n(x),$$

где $A_n(x) = 1/n$; $B_n(x) = x^n$ (заметим, что значение x может быть отрицательным).

Алгоритм представлен на рис. 5.

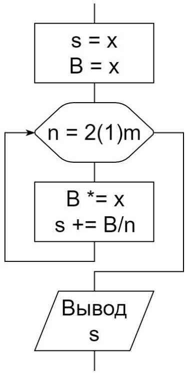

Рис. 5. Пример вычисления конечной суммы с помощью алгоритма №3

В общем случае применение одного из трех рассмотренных алгоритмов диктуется целесообразностью при решении той или иной задачи, но также зависит от личных предпочтений программиста.

При вычислении **конечного произведения**

$$P = \prod_{n=0}^{m} [1 + t_n(x)]$$

можно использовать модификацию того или иного алгоритма вычисления конечной суммы.

<u>**Пример 2.**</u> Составим программу вычисления значений функции:

$$z = \begin{cases} \dfrac{x+1}{2} \sum\limits_{n=1}^{10} \left(\dfrac{x}{n}\right)^n, & x \le 2; \\ \dfrac{sinx + cosx}{2 + sinx} \prod\limits_{n=0}^{15} \left(1 + \dfrac{x}{n+2}\right), & x > 2. \end{cases}$$

В зависимости от значения переменной x реализуется вычисление суммы или произведения (рис. 6).

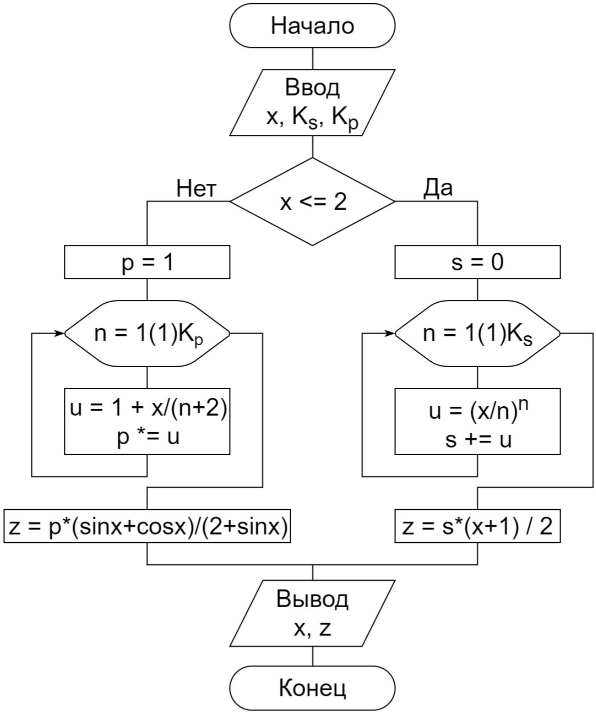

Рис. 6. Алгоритм вычисления суммы или произведения

Вычисление суммы целесообразно реализовать с помощью оператора цикла с параметром *n*. В теле цикла необходимо вычислить значение очередного слагаемого $u_n = (x/n)^n$ при текущем *n* и осуществить накопление суммы по формуле $S_n = S_{n-1} + u_n$ (если $x > 0$). Подобные операции требуется выполнить для $n = 1(1)K_s$. Так как нет необходимости запоминать значения всех слагаемых $u_1, u_2, ..., u_{ks}$ и конечных сумм $S_1, S_2, ..., S_{ks}$, то в качестве S_n и u_n можно использовать скалярные переменные *S* и *u*. При этом накопление суммы можно реализовать с помощью операции *S += u*. Перед выполнением цикла значение переменной *S* должно быть нулевым.

Вычисление произведения организуем с помощью аналогичной циклической структуры с параметром. В данном случае необходимо вычислять сомножитель $u = 1 + x/(n+2)$ и произведение по формуле *P *= u*. Перед выполнением цикла переменной *P* должно быть присвоено значение 1.

Для обеспечения большей универсальности алгоритма обозначим предел суммирования через K_s, а предел произведения через K_p и обеспечим их ввод в программе в качестве исходных данных.

Программа имеет следующий вид:

```
from math import *

# Цель: Вычисление сложной функции (конечная сумма и
#       произведение).
# Переменные:
#      z - значение функции;
#      x - аргумент функции;
#      S - сумма, P - произведение
#      n - переменная суммирования и произведения;
#      u - слагаемое (множитель);
#      Ks - число слагаемых; Kp - число сомножителей.
# Программист: Степанов Н.Н.
# Дата написания: 25.02.2017.

print("Введите исходные данные:")

print("x = ", end='')
x = float(input())
print("Ks = ", end='')
Ks = int(input())
print("Kp = ", end='')
Kp = int(input())

print("Вы ввели:")
print("X = %.2f   Ks = %d   Kp = %d" % (x, Ks, Kp))

if x <= 2:
    S = 0
    for n in range(1, Ks + 1):
        u = exp(n * log(x / n))
        S += u
    z = S * (x + 1) / 2
else:
    P = 1
    for n in range(1, Kp + 1):
        P *= 1 + x / (n + 2)
```

```
        T = sin(x)
        z = (T + cos(x) * P / (2 + T))

print("Результат:")
print("X = %.3f   Y= %.3f" % (x, z))
```

Оператор continue

Тело цикла возможно разделить на две части и вторую часть (в виде совокупности команд, расположенных в конце тела цикла) исключать из процесса выполнения в зависимости от некоторого условия. Оператор `continue` начинает следующее повторение тела цикла, минуя команды, стоящие после него. Оператор continue можно использовать в операторах цикла `while` или `for`. Например, программа

```
for i in range(1, 6):
    if i == 3:
        continue
    print("2^%d = %d " % (i, 2**i))
```

позволяет получить результат:

```
2^1 = 2
2^2 = 4
2^4 = 16
2^5 = 32
```

Оператор break

Существует возможность досрочного выхода из цикла в случае выполнения некоторого условия. С этой целью в операторах `while` и `for` используют оператор `break`.

При выполнении программы

```
for i in range(1, 6):
    if i >= 3:
        break
    print("2^%d = %d " % (i, 2**i))
```

выводятся следующие значения:

```
2^1 = 2
2^2 = 4
```

Конструкция else

Слово `else`, примененное в цикле `for` или `while`, проверяет, был ли произведен выход из цикла оператором `break` или же "естественным" образом. Блок инструкций внутри `else` выполнится только в том случае, если выход из цикла произошел без помощи `break`.

В качестве примера ниже приведена программа, демонстрирующая работу конструкции `else`.

```
a = [1, 2, 3, 4]
for i in a:
    if i < 0:
        break
else:
    print("В списке " + str(a) + " нет отрицательных значений!")
```

После выполнения программы на экране дисплея выведутся следующие результаты:

```
В списке [1, 2, 3, 4] нет отрицательных значений!
```

Контрольные вопросы

1. Какие основные правила организации цикла с параметром?
2. Какова схема табулирования функции с использованием оператора цикла с заголовком?
3. Какие ограничения существуют на тип параметра в заголовке оператора `for`?
4. Какие алгоритмы можно использовать при вычислении конечных сумм?
5. В чем отличие алгоритмов нахождения конечной суммы и конечного произведения?
6. Что позволяет выполнить использование оператора `continue`?
7. Каким образом задается закон изменения параметра цикла с помощью функции `range`?
8. Каково назначение оператора `break`?
9. Для чего используется конструкция `else`?

Задания

Для задания в соответствии со своим вариантом составить алгоритм и написать программу, имеющие структуру цикла с заголовком и осуществляющие нахождение указанного значения конечной суммы или конечного произведения.

Варианты задания:

1. Вычислить сумму для $x = 1,2$:

$$S = \sum_{n=0}^{10} \frac{x^{4n+1}}{4n+1}.$$

2. Вычислить сумму:

$$S = \sum_{n=1}^{20} (a^n + 1) \ln x; \quad a = \begin{cases} 0,5, & \text{если } n \geq 12 \text{ и } x \geq 3,5; \\ 7,5 & \text{в остальных случаях.} \end{cases}$$

3. Вычислить произведение для $x = 2,3$:

$$P = \prod_{i=1}^{10} \frac{2x+1}{(2i)^2 + 1}.$$

4. Вычислить сумму:

$$S = \sum_{n=1}^{10} n^2 + \sum_{n=1}^{12} n^3.$$

5. Вычислить произведение при *a = 2*:

$$P = \prod_{i=1}^{10} \frac{i(i+a)}{i^2+a^2} + \prod_{k=2}^{4} \frac{k^3}{k+a}.$$

6. Вычислить:

$$W = \begin{cases} \sum_{k=1}^{10} x^k sin\frac{k\pi}{4}, & x \geq a; \\ \prod_{m=1}^{5} (a^m - x^m), & x < a. \end{cases}$$

Для контрольного просчета принять *x = 7,5; a = 1,7.*

7. Вычислить:

$$\lambda = (l \cdot k)!, \qquad l = \begin{cases} 2, & \text{если } k \text{ четное;} \\ 1, & \text{если } k \text{ нечетное.} \end{cases}$$

$$\text{причем } n! = 1 \cdot 2 \cdot \dots \cdot n = \prod_{m=1}^{n} m$$

Для контрольного просчета принять *k = 5*.

8. Вычислить:

$$y = \frac{\sin x + 2}{3 + \cos x} \sum_{n=0}^{20} a x^n; \quad a = \begin{cases} 2n, & x \leq 0{,}5; \\ \frac{n}{2}, & x > 0{,}5. \end{cases}$$

Для контрольного просчета принять *x = 0,5; a = 1,7.*

9. Вычислить:

$$y = \frac{ax^2}{\sqrt{x+a}}; \quad a = \begin{cases} \sum_{k=1}^{12} \frac{2kx}{x+k^2}, & x < 1; \\ 1, & x \geq 1. \end{cases}$$

Для контрольного просчета принять *x = 7,5; a = 0,7.*

10. Вычислить:

$$z = \sum_{k=0}^{10} \ln x \cdot \sin k(x-a); \quad a = \begin{cases} \frac{\pi}{4}, & x \leq 1; \\ \pi, & x > 1. \end{cases}$$

Для контрольного просчета принять *x = 1,5; a = 2,7.*

11. Вычислить:

$$z = \begin{cases} \sum_{n=1}^{10} \frac{a^2}{a^n - 5}, & a < 4; \\ \frac{a+1}{a} \prod_{n=1}^{8} \frac{a-1}{n}, & a \geq 4. \end{cases}$$

Для контрольного просчета принять *a = 1,7*.

12. Вычислить:

$$P = (x \cdot t)!; \quad n! = 1 \cdot 2 \cdot \dots \cdot n; \quad t = \begin{cases} 1{,}5, \text{если } x - \text{ четное}; \\ 2, \text{если } x - \text{ нечетное}. \end{cases}$$

Для контрольного просчета принять *x = 3*.

13. Вычислить:

$$z = \begin{cases} \ln(1-x), & x \leq 0; \\ \ln(1+x), & x > 0. \end{cases}$$

Для вычисления ln(1 − x) воспользоваться равенством

$$\ln(1-x) = -\sum_{n=1}^{50} \frac{x^n}{n}.$$

Для контрольного просчета принять *x = 0,5*.

14. Вычислить:

$$F = \frac{a+x}{3} \sum_{n=0}^{6} (x+a)^{n/2}.$$

Для контрольного просчета принять *x = 7,5; a = 1,7*.

15. Вычислить:

$$z = \begin{cases} a \cdot \ln x, & x \geq a; \\ x \cdot \ln a, & x < a. \end{cases}$$

Для вычисления ln x воспользоваться равенством

$$\ln x \approx 2 \sum_{n=0}^{10} \frac{(x-1)^{2n-1}}{(2n+1)(x+1)^{2n+1}}.$$

Для контрольного просчета принять *x = 1,5; a = 1,7*.

16. Вычислить произведение:

$$y = \prod_{n=1}^{8} \left(p - \frac{x^n}{2n+1}\right); \quad p = \begin{cases} 1, & n \leq 5; \\ 2, & x > 5. \end{cases}$$

Для контрольного просчета принять *x = 0,7*.

17. Вычислить сумму:

$$t = a\sum_{i=1}^{10}\frac{i}{i+a^i}.$$

Для контрольного просчета принять *a = 1,7*.

18. Вычислить:

$$y = \begin{cases} \sum_{k=1}^{8} a^k x^k, & a \le x; \\ \prod_{n=1}^{8} (a^k - x^k), & a > x. \end{cases}$$

Для контрольного просчета принять *x = 2,5; a = 1,7*.

19. Вычислить:

$$z = \begin{cases} \prod_{n=1}^{8} \left(\frac{x}{2}\right)^n, & x \le 2; \\ \sum_{n=0}^{5} (1 + xn), & x > 2. \end{cases}$$

Для контрольного просчета принять *x = 2,3*.

20. Вычислить:

$$F = e^{-x} - x!; \quad x! = \prod_{n=1}^{x} n; \quad 0! = 1.$$

Для контрольного просчета принять *x = 5*.

21. Вычислить произведение:

$$\lambda = \prod_{n=1}^{5}\left(\frac{1}{n} + \mathrm{a} \cdot \sin x\right); \quad a = \begin{cases} 1, & x \ge 0; \\ -1, & x < 0. \end{cases}$$

Для контрольного просчета принять *x = 1,5*.

22. Вычислить произведение:

$$y = \prod_{n=1}^{5}\left(n + \frac{\sin x}{n}\right).$$

Для контрольного просчета принять *x = 0,7*.

23. Вычислить сумму:

$$x = \sum_{n=0}^{4} (k+n) a^{k+n}.$$

Для контрольного просчета принять *k = 5, a = 1,7*.

Литература

1. Python на примерах. Практический курс по программированию. / А.Н. Васильев. – СПб.: Наука и Техника, 2016. – 432 с.
2. http://pythonicway.com
3. https://pythonworld.ru
4. http://www.python-course.eu/python3_course.php

Содержание