

1.4. Математические вычисления, модуль `math`

В программах можно выполнять достаточно сложные вычисления, в которых используются математические функции, такие как логарифм, экспонента, синус, косинус и т.д. Для вычисления этих и многих других математических функций в Python предусмотрена библиотека **`math`** (или по-другому модуль **`math`**). Библиотека является встроенным модулем Python, поэтому для ее использования никакой дополнительной установки не требуется.

1.4.1. Модуль `math`

Модуль **`math`** содержит множество функций для работы с числовыми данными, а также математические константы. В таблице 7 приведены некоторые из них.

Таблица 7 – Функции модуля `math`

Функция	Описание	Пример
<code>ceil(x)</code>	Округление до ближайшего большего числа	<code>ceil(4.2)=5</code> <code>ceil(-5.8)=-5</code>
<code>floor(x)</code>	Округление до ближайшего меньшего числа	<code>floor(4.2)=4</code> <code>floor(-5.8)=-6</code>
<code>exp(x)</code>	Экспонента от <code>x</code> (<code>e^x</code>)	<code>exp(3)=20.08...</code>
<code>log(x, base)</code>	Логарифм <code>x</code> по основанию <code>base</code> , если <code>base</code> не указан, вычисляется натуральный логарифм	<code>log(125, 5)=3.0</code>
<code>log10(x)</code>	Логарифм <code>x</code> по основанию 10	<code>log10(100)=2.0</code> <code>log10(0.01)=-2.0</code>
<code>log2(x)</code>	Логарифм <code>x</code> по основанию 2	<code>log2(64)=6.0</code> <code>log2(0.125)=-3.0</code>
<code>pow(x, y)</code>	<code>x^y</code>	<code>pow(3, 4)=81.0</code>
<code>sqrt(x)</code>	Квадратный корень из <code>x</code>	<code>sqrt(4)=2.0</code> <code>sqrt(2)=1.41...</code>
<code>acos(x)</code>	Арккосинус <code>x</code> (<code>x</code> в интервале от -1 до 1), Результат – значение в радианах	<code>acos(0)=1.57...</code> <code>acos(1)=0.0</code>
<code>asin(x)</code>	Арксинус <code>x</code> (<code>x</code> в интервале от -1 до 1), Результат – значение в радианах	<code>asin(0)=0.0</code> <code>asin(1)=1.57...</code>
<code>atan(x)</code>	Арктангенс <code>x</code> , Результат – значение в радианах	<code>atan(0)=0.0</code> <code>atan(1)=0.78...</code>
<code>cos(x)</code>	Косинус <code>x</code> (<code>x</code> задается в радианах)	<code>cos(0)=1.0</code>
<code>sin(x)</code>	Синус <code>x</code> (<code>x</code> задается в радианах)	<code>sin(0)=0.0</code>
<code>tan(x)</code>	Тангенс <code>x</code> (<code>x</code> задается в радианах)	<code>tan(0)=0.0</code>
<code>degrees(x)</code>	Конвертирует радианы в градусы	<code>degrees(3) = 171.8...</code>
<code>radians(x)</code>	Конвертирует градусы в радианы	<code>radians(180)=3.14...</code>
<code>pi</code>	<code>pi = 3.1415926...</code>	
<code>e</code>	<code>e = 2.718281...</code>	
<code>inf</code>	<code>∞</code>	

Практически все функции модуля **math** возвращают значения в вещественном формате. В некоторых случаях может получиться такой результат:

```
cos(1.5707963) = 2.6794896585e-08
```

Число вида $2.6794896585e-08$ является записью вещественного числа в экспоненциальной форме, переписать его можно так:

$$2.6794896585e-08 = 2.6794896585 \cdot 10^{-8} = 0.000000026794896585,$$

то есть это практически ноль или, точнее, число равно нулю с точностью 7 знаков после запятой.

В программе, где предполагается использование модуля, применяется оператор:

```
import math
```

Этот оператор должен располагаться до первого обращения к функциям модуля. Как правило, все импортируемые модули перечисляются в начале программы.

В выражениях при использовании функций из этого модуля перед именем функции вставляется префикс **math** и точка. В таблице 8 приведены примеры выражений, для вычисления которых необходимы функции из модуля **math**.

Таблица 8 – Примеры описания выражений с явным указанием имени модуля

Выражение	Код программы
$\sin(x) + \cos(x)$	<code>math.sin(x) + math.cos(x)</code>
$\log_5(x \cdot e^{2 \cdot x})$	<code>math.log(x * math.exp(2 * x), 5)</code>
$\sqrt{b^2 - 4 \cdot a^2}$	<code>math.sqrt(b ** 2 - 4 * a ** 2)</code> <code>math.sqrt(math.pow(b, 2) - 4 * math.pow(a, 2))</code>
$\sin(40^\circ)$	<code>math.sin(math.radians(40))</code>
π	<code>math.pi</code>

В случае, если точно известно, какие функции из модуля понадобятся в программе, можно их перечислить при подключении модуля и уже в тексте программы не использовать префикс **math**:

```
from math import функция_1, функция_2, ...
```

В таблице 9 приведены примеры описания выражений в программе, если все необходимые функции перечислены при импорте:

```
from math import sin, cos, log, exp, pow, radians, pi
```

Таблица 9 – Примеры описания выражений без явного указания имени модуля

Выражение	Код программы
$\sin(x) + \cos(x)$	<code>sin(x) + cos(x)</code>
$\log_5(x \cdot e^{2 \cdot x})$	<code>log(x * exp(2 * x), 5)</code>
$\sqrt{b^2 - 4 \cdot a^2}$	<code>sqrt(b ** 2 - 4 * a ** 2)</code> <code>sqrt(pow(b, 2) - 4 * pow(a, 2))</code>
$\sin(40^\circ)$	<code>sin(radians(40))</code>
π	<code>pi</code>

Задания для самопроверки

1. Какие функции из модуля **math** записаны неверно? Исправьте ошибки в неверных выражениях.

```
math.log(10, 2)
math.exp(3, 5)
math.sin(-16)
math.degrees(16)
math sin(3)
math.radians(3)
```

2. Используя функции модуля **math**, запишите следующие выражения:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \quad \pi r^2.$$

3. Написать программу, которая вычисляет расстояние между двумя точками на плоскости $A(x_0, y_0)$ и $B(x_1, y_1)$. Координаты точек ввести с клавиатуры, результат округлить до 4 знаков после запятой. Формула для вычисления расстояния:

$$r = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

Проверить программу для следующих входных данных:

Тест 1

$x_0 = -4.5$

$y_0 = 2.7$

$x_1 = 1$

$y_1 = 3.1$

Результат: 5.5145

Тест 2

$x_0 = 1$

$y_0 = 1$

$x_1 = 2$

$y_1 = 2$

Результат: 1.4142

1.4.2. Рекомендации по записи сложных выражений

Рекомендуется **перечислить** все используемые **функции** для вычисления выражения при подключении модуля **math**, чтобы запись формул была короче и понятнее.

Пример. Вычислить значение выражения:

$$y = \sqrt{e^{\sin(x)} + \ln(2 + \cos(x))}$$

при $x = 4.5$.

Код:

Вариант 1

```
import math
x = 4.5
y = math.sqrt(math.exp(math.sin(x)) + math.log(2 + math.cos(x)))
print(y)
```

Вариант 2

```
from math import sqrt, exp, sin, log, cos
x = 4.5
y = sqrt(exp(sin(x)) + log(2 + cos(x)))
print(y)
```

Результат:

0.9787799339565816

Сравните два варианта программы, обе программы выдают одинаковый результат, но способ с перечислением функций выглядит более удобочитаемым ~~читабельным~~.

При записи сложных математических выражений можно допустить ошибки, например, забыть поставить скобку или пропустить операцию. Поэтому рекомендуется **разбить выражение на части**, занести их в отдельные переменные, а затем вычислить полное выражение.

Пример. Вычислить значение выражения:

$$\ln \left(\left| y - \sqrt{\sin^2 \left(x + \frac{\pi}{3} \right)} \right| + \cos \left(tg^2 \left(\frac{1}{\sqrt[3]{z+1}} \right) \right) \right)$$

при $x = 3.6$, $y = -4.1$, $z = 5$.

Решение

Из данного выражения целесообразно выделить две части и вычислить их отдельно (рисунок 1), сохранив результаты в переменные **eq_1** и **eq_2**, а затем получить результат, используя эти переменные.

$$\ln\left(\left|y - \sqrt{\sin^2\left(x + \frac{\pi}{3}\right)}\right|\right) + \cos\left(\operatorname{tg}^2\left(\frac{1}{\sqrt[3]{z+1}}\right)\right)$$

Рисунок 1. Пример выделения частей выражения

Код:

```
from math import log, sqrt, sin, pi, cos, tan

x = 3.6
y = -4.1
z = 5
eq_1 = sqrt(sin(x + pi / 3) ** 2)
eq_2 = cos(tan(1/((z+1) ** (1/3))) ** 2)
eq = log(abs(y - eq_1) + eq_2)
print("eq =", eq)
```

Результат:

```
eq = 1.796391214837781
```

Задание для самопроверки

Написать программу для вычисления следующего выражения, значения **x** и **y** ввести с клавиатуры, результат округлить до 5 знаков после запятой:

$$z = \frac{\arcsin\left(\cos\left(x + \frac{\sqrt{3}}{2}\pi\right)\right) + 1.2\sqrt{2 - \cos^2(y)}}{x^2 + y^2 + 1}.$$

Проверить программу для следующих входных данных:

Тест 1

$x = -2.3$

$y = 8.72$

Результат: 0.03134

Тест 2

$x = -7.1$

$y = 12.8$

Результат: 0.00417

1.5. Обработка исключительных ситуаций

При вычислении выражений для некоторых значений могут возникать ошибки при выполнении программы, такие как деление на ноль, переполнение и пр. Например, при вычислении выражения:

$$y = 1 / x$$

если **x** будет иметь значение 0, то возникнет ошибка «Деление на ноль» и программа закончит свое выполнение.

Подобные ошибки необходимо обработать при реализации программы. Можно найти область определения выражения (в нашем случае **x** не должно равняться 0), а в программу включить условный оператор, в котором проверить значение **x**, если значение равно 0 – вывести ошибку, в противном случае выполнить вычисление:

```

if x == 0:
    print("Деление на ноль")
else:
    y = 1 / x

```

Однако, если выражение сложное, описать область ее определения часто не представляется возможным. Тогда можно использовать блок обработки исключений **try ... except**. С его помощью осуществляется отслеживание и обработка ошибок, возникающих в процессе вычисления выражений. Блок обработки исключений состоит из двух частей:

- раздел **try**, где размещаются операторы, в которых может возникнуть ошибка;
- раздел **except**, в нем размещаются операторы, выполняемые при возникновении ошибок в разделе **try**, также в нем можно реализовать различные действия для различных типов ошибок.

Выполняется блок обработки исключений следующим образом:

- вычисляются операторы, размещенные в разделе **try**, если в каком-то из них возникает ошибка, управление передается блоку **except**, в противном случае выполняется первый оператор после блока обработки исключений;
- если управление передается в раздел **except**, то выполняются операторы в этом блоке, затем управление передается первому оператору после блока обработки исключений.

С использованием блока обработки исключений наш пример будет выглядеть следующим образом:

```

x = int(input("x = "))
try:
    y = 1 / x
except:
    print("Деление на ноль")
print("y =", y)

```

Протестируем нашу программу на различных входных данных:

Тест 1

```

x = 6
y = 0.16666666666666666

```

Тест 2

```

x = 0
Деление на ноль
Traceback (most recent call last):
  File "test.py", line 6, in <module>
    print("y =", y)
NameError: name 'y' is not defined

```

Программа выдает ошибку при вводе значения 0, так как при таком использовании блока обработки исключений после него осуществляется вывод вычисленного значения **y**, которое не определено.

Исправить ошибку можно, поместив вывод в разделе **try**:

```

x = int(input("x = "))
try:
    y = 1 / x
    print("y =", y)
except:
    print("Деление на ноль")

```

Тогда оба теста проходят.

Тест 1

```

x = 6
y = 0.16666666666666666

```

Тест 2

```

x = 0
Деление на ноль

```

Такой вариант подходит, если в дальнейшем переменную **y** нигде не нужно использовать. Если же результат (бесконечность), который получается при делении на ноль, важен для вычислений, можно занести в переменную константу **inf**, которая определена в модуле **math**. Наша программа в этом случае будет выглядеть следующим образом:

```
import math
x = int(input("x = "))
try:
    y = 1 / x
except:
    y = math.inf
print("y =", y)
```

Протестируем программу на различных входных данных:

Тест 1

```
x = 6
y = 0.16666666666666666
```

Тест 2

```
x = 0
y = inf
```

Задание для самопроверки

Написать программу для вычисления следующего выражения, значения **x** и **y** ввести с клавиатуры. В случае возникновения ошибок вычисления вывести **inf**.

$$z = \frac{\arccos(x) + 1.2\sqrt{y}}{x^2 - (y + 1)^2}$$

Проверить программу для следующих входных данных:

Тест 1

```
x = 0.5
y = 6
z = -0.08177610753920843
```

Тест 2

```
x = 0.5
y = -5
z = inf
```

Тест 3

```
x = 1
y = 0
z = inf
```

Тест 4

```
x = 2
y = 1
z = inf
```