Python. Начала программирования

Линейный алгоритм

В качестве примера разработки линейного алгоритма рассмотрим задачу вычисления значения арифметического выражения $ax^2 + bx + c$.

Представим первоначальное описание алгоритма на естественном языке:

1. ***Начало***
2. ***Положить a = 1***
3. ***Положить b = 2***
4. ***Положить c = 3***
5. ***Ввести значение x***
6. ***Вывести значение a * x * x + b * x + c***
7. ***Конец***

Представим альтернативную версию описания алгоритма на естественном языке:

```
begin
   a = 1
   b = 2
   c = 3
   input x
   print a * x * x + b * x + c
end
```

Представим программу для целочисленной арифметики:

```
a = 1
b = 2
c = 3
x = int(input('? '))
print(a * x * x + b * x + c)
```

Результат работы программы:

```
? 4
27
```

Представим программу для вещественной арифметики:

```
a = 1
b = 2
c = 3
x = float(input('? '))
print(a * x * x + b * x + c)
```

Результат работы программы:

```
? 0.5
4.25
```

Для вычисления выражения ***x * x*** обратимся к оператору возведения в степень.
Представим программу для целочисленной арифметики:

```
a = 1
b = 2
c = 3
x = int(input('? '))
print(a * x ** 2 + b * x + c)
```

Результат работы программы:

```
? 4
27
```

Представим программу для вещественной арифметики:

```
a = 1
b = 2
c = 3
x = float(input('? '))
print(a * x ** 2 + b * x + c)
```

Результат работы программы:

```
? 0.5
4.25
```

Обратимся к стандартной математической функции ***pow(x, y)*** из модуля ***math*** для вычисления выражения ***x*** в степени ***y***. Все функции этого модуля оперируют целыми числами и числами с плавающей точкой, и они возвращают число с плавающей точкой.
Представим программу для целочисленной арифметики:

```
import math
a = 1
b = 2
c = 3
x = int(input('? '))
print(a * int(math.pow(x, 2)) + b * x + c)
```

Результат работы программы:

```
? 4
27
```

Представим программу для вещественной арифметики:

```
import math
a = 1
b = 2
c = 3
x = float(input('? '))
print(a * math.pow(x, 2) + b * x + c)
```

Результат работы программы:

```
? 0.5
4.25
```

Отметим, что с помощью инструкции ***import*** модули по умолчанию загружаются из стандартной библиотеки Python.

Напомним, что когда инструкция ***import*** впервые загружает модуль, она выполняет следующие три операции:

1. Создаёт новое пространство имён, которое будет служить контейнером для всех объектов, объявленных в модуле.
2. Выполняет программный код объекта, объявленного в модуле, внутри вновь созданного пространства имён.
3. Создаёт в вызывающей программе имя, ссылающееся на пространство имён модуля. Это имя совпадает с именем модуля и используется для доступа к указанному объекту модуля.

Для загрузки отдельных определений из модуля в текущее пространство имён используется инструкция ***from***. По своей функциональности инструкция ***from*** идентична инструкции ***import***, за исключением того, что вместо создания имени, ссылающегося на вновь созданное пространство имён модуля, она помещает ссылки на один или более объектов, объявленных в модуле, в текущее пространство имён.

Представим программу для целочисленной арифметики:

```
from math import pow
a = 1
b = 2
c = 3
x = int(input('? '))
print(a * pow(x, 2) + b * x + c)
```

Результат работы программы:

```
? 4
27.0
```

Как видим, получили ожидаемый результат вещественного типа.

Обратимся теперь к встроенной функции ***pow(x, y, z)***, которая возвращает результат выражения ***x ** y*** при отсутствии третьего аргумента *z*, а при его наличии – результат выражения ***(x ** y) % z***, при этом все три аргумента ***x, y*** и ***z*** должны быть целыми числами, а аргумент ***y*** в этом случае должен быть неотрицательным.

Представим программу для целочисленной арифметики:

```
a = 1
b = 2
c = 3
x = int(input('? '))
print(a * pow(x, 2) + b * x + c)
```

Результат работы программы:

```
? 4
27
```

Как видим, возвращаемое значение для функции ***pow(x, 2)*** здесь неявно приведено к целочисленному типу, поскольку оба аргумента являются целочисленными объектами.

Представим программу для вещественной арифметики:

```
a = 1
b = 2
c = 3
x = float(input('? '))
print(a * pow(x, 2) + b * x + c)
```

Результат работы программы:

```
? 0.5
4.25
```

Как видим, возвращаемое значение для функции ***pow(x, 2)*** здесь уже приведено к вещественному типу, поскольку первый аргумент является вещественным объектом.

Теперь обратимся к механизму защиты данных при вводе объектов встроенных типов, основанному на перехвате исключений, возбуждённых программным кодом.

Поначалу рассмотрим поведение Python Shell в случае ввода некорректного значения целочисленного объекта, загрузив программу для целочисленной арифметики:

```
? a
Traceback (most recent call last):
  File "C:\Users\user\Desktop\Python\problem1.01.py",
  line 4, in <module>
  x = int(input('? '))
ValueError: invalid literal for int() with base 10: 'a'
```

Как видим, при попытке ввести некорректное значение для целочисленного объекта ***x*** будет возбуждено встроенное исключение ***ValueError***.

Теперь перехватим исключение ***ValueError*** с помощью инструкции ***try*** с блоком ***except*** и блоком ***else***.

Представим программу для целочисленной арифметики:

```
a = 1
b = 2
c = 3
try :
  x = int(input('? '))
except ValueError as message :
  print(message)
else :
  print(a * x * x + b * x + c)
```

Результат работы программы (в случае корректного ввода):

```
? 4
27
```

Результат работы программы (в случае некорректного ввода):

```
? a
invalid literal for int() with base 10: 'a'
```

Напомним, что все составные инструкции (то есть инструкции, которые включают вложенные в них инструкции) записываются в соответствии с одним и тем же шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок инструкций, обычно с отступом под строкой основной инструкции. Величина отступа наглядно показывает, какой основной инструкции принадлежит её вложенный блок инструкций.

Теперь рассмотрим поведение Python Shell в случае ввода некорректного значения вещественного объекта, загрузив программу для вещественной арифметики:

```
? a
Traceback (most recent call last):
  File "C:\Users\user\Desktop\Python\problem1.02.py",
  line 4, in <module>
  x = float(input('? '))
ValueError: could not convert string to float: 'a'
```

Как видим, при попытке ввести некорректное значение для вещественного объекта ***x*** будет возбуждено встроенное исключение ***ValueError***.

Теперь перехватим исключение ***ValueError*** с помощью инструкции ***try*** с блоком ***except*** и блоком ***else***.

Представим программу для вещественной арифметики:

```
a = 1
b = 2
c = 3
try :
  x = float(input('? '))
except ValueError as message :
  print(message)
```

```
else :
  print(a * x * x + b * x + c)
```

Результат работы программы (в случае корректного ввода):

```
? 0.5
4.25
```

Результат работы программы (в случае некорректного ввода):

```
? a
could not convert string to float: 'a'
```