Python. Начала программирования

Условные алгоритмы

В качестве примера разработки условного алгоритма с двумя ветвями рассмотрим задачу вычисления значения модуля числа $|x| = \begin{cases} x, & \text{если } x \geq 0 \\ -x, & \text{если } x < 0 \end{cases}$

Представим первоначальное описание алгоритма на естественном языке:

1. Начало
2. Ввести значение x
3. Если $x < 0$
то
3.1. Вывести значение $-x$
иначе
3.2. Вывести значение x
4. Конец

Представим альтернативную версию описания алгоритма на естественном языке:

```
begin
  input x
  if x < 0
  then
        print − x
  else
        print x
end
```

Представим программу для целочисленной арифметики:

```
x = int(input('? '))
if x < 0 :
  print(-x)
else :
  print(x)
```

Результат работы программы:

```
? 7
7
```

Результат работы программы:

```
? -7
7
```

Представим программу для вещественной арифметики:

```
x = float(input('? '))
if x < 0 :
  print(-x)
else :
  print(x)
```

Результат работы программы:

```
? 2.7
2.7
```

Результат работы программы:

```
? -2.7
2.7
```

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

Представим программу для целочисленной арифметики:

```
try :
  x = int(input('? '))
except ValueError as message :
  print(message)
else :
  if x < 0 :
    print(-x)
  else :
    print(x)
```

Результат работы программы (в случае корректного ввода):

```
? 7
7
```

Результат работы программы (в случае некорректного ввода):

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

Представим программу для вещественной арифметики:

```
try :
  x = float(input('? '))
except ValueError as message :
  print(message)
```

```
else :
  if x < 0 :
    print(-x)
  else :
    print(x)
```

Результат работы программы (в случае корректного ввода):

```
? 2.7
2.7
```

Результат работы программы (в случае некорректного ввода):

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

В качестве примера разработки условного алгоритма с множеством ветвей рассмотрим задачу вычисления значения функции $y = \begin{cases} a + x & \text{при } a = 1 \\ (a + x)^2 & \text{при } a = 2 \\ (a + x)^3 & \text{при } a = 3 \end{cases}$

Представим первоначальное описание алгоритма на естественном языке:

1. ***Начало***
2. ***Ввести значение x***
3. ***Ввести значение a***
4. ***По значению переключателя a выбрать***
   Ветвь 1 :
   4.1. Вывести значение a + x
   Ветвь 2 :
   4.2. Вывести значение (a + x) * (a + x)
   Ветвь 3 :
   4.3. Вывести значение (a + x) * (a + x) * (a + x)
   иначе
   4.4. Вывести значение 'Ошибка!'
5. ***Конец***

Представим альтернативные версии описания алгоритма на естественном языке:

```
begin
  input x
  input a
  switch a
    case 1 :
      print a + x
    case 2 :
      print (a + x) * (a + x)
    case 3 :
      print (a + x) * (a + x) * (a + x)
```

```
    else
      print ‘Error!‘
end
```

```
begin
  input x
  input a
  if a = 1
  then
    print a + x
  else
    if a = 2
    then
      print (a + x) * (a + x)
    else
      if a = 3
      then
        print (a + x) * (a + x) * (a + x)
      else
        print ‘Error!‘
end
```

Как видим, наряду с переключателем для реализации множественного выбора можно воспользоваться и вложенными условными инструкциями ***if***.

Заметим, что в рамках структурного подхода предпочтение следует отдавать именно переключателю при наличии соответствующей инструкции в языке программирования.

Представим программу для вещественной арифметики, опираясь только на вторую версию алгоритма, поскольку в языке Python отсутствует инструкция ***switch***:

```
x = float(input('x? '))
a = int(input('a? '))
if a == 1 :
  print(a + x)
else :
  if a == 2 :
    print((a + x) ** 2)
  else :
    if a == 3 :
      print((a + x) ** 3)
    else :
      print('Error!')
```

Результат работы программы:

```
x? 5
a? 0
Error!
```

Результат работы программы:

```
x? 5
a? 1
6.0
```

Результат работы программы:

```
x? 5
a? 2
49.0
```

Результат работы программы:

```
x? 5
a? 3
512.0
```

Известно, что в условной инструкции ***if***, которая является составной инструкцией, могут присутствовать одна или более необязательных частей ***elif*** (“***else if***”) и, наконец, необязательная часть ***else***. Условные выражения как самой инструкции ***if***, так и её частей ***elif*** предназначены для обработки каждой альтернативной ветви множественного выбора.

Представим альтернативную версию этой программы с использованием частей ***elif*** в условной инструкции ***if***:

```
x = float(input('x? '))
a = int(input('a? '))
if a == 1 :
  print(a + x)
elif a == 2 :
  print((a + x) ** 2)
elif a == 3 :
  print((a + x) ** 3)
else :
  print('Error!')
```

Результат работы программы:

```
x? 5
a? 0
Error!
```

Результат работы программы:

```
x? 5
a? 1
6.0
```

Результат работы программы:

```
x? 5
a? 2
49.0
```

Результат работы программы:

```
x? 5
a? 3
512.0
```

Как видим, каждая часть ***elif*** по своему действию похожа на ***case***-ветвь инструкции ***switch***. Часть ***else*** предназначена для обработки ситуации, когда не будет найдено ни одного совпадения.

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

Представим версию этой программы с использованием инструкции ***try*** с блоком ***except*** и блоком ***else***:

```
try :
  x = float(input('x? '))
  a = int(input('a? '))
except ValueError as message :
  print(message)
else :
  if a == 1 :
    print(a + x)
  elif a == 2 :
    print((a + x) ** 2)
  elif a == 3 :
    print((a + x) ** 3)
  else :
    print('Error!')
```

Результат работы программы (в случае корректного ввода):

```
x? 5
a? 0
Error!
```

Результат работы программы (в случае корректного ввода):

```
x? 5
a? 1
6.0
```

Результат работы программы (в случае некорректного ввода):

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

Результат работы программы (в случае некорректного ввода):

```
x? 5
a? a
invalid literal for int() with base 10: 'a'
```