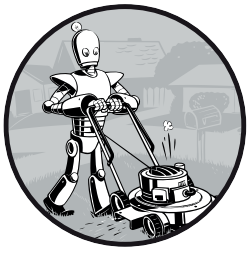
8

ПРОВЕРКА ВВОДА



В программе необходимо проверять корректность значений, вводимых пользователем с помощью той же функции `input()`. В частности, если пользователь должен указать свой возраст, то программа не должна принимать бессмысленные ответы, например отрицательные числа (недопустимый диапазон целых чисел) или слова (неправильный тип данных). Проверка ввода также позволяет предотвратить ошибки. Если вы реализуете функцию `withdrawFromAccount()`, которая содержит аргумент для снятия суммы со счета, то убедитесь, что эта сумма положительна. При попытке снять со счета отрицательную сумму функция в конечном итоге добавит деньги на счет!

Как правило, мы выполняем проверку ввода, неоднократно запрашивая данные у пользователя, пока он не введет корректный текст, как в следующем примере.

```
while True:
    print('Укажите ваш возраст:')
    age = input()
    try:
        age = int(age)
    except:
        print('Пожалуйста, введите цифры.')
        continue
    if age < 1:
        print('Пожалуйста, введите положительное число.')
        continue
    break

print(f'Вам {age} лет.')
```

После запуска программы результат будет выглядеть примерно так.

```
Укажите ваш возраст:
тридцать
Пожалуйста, введите цифры.
Укажите ваш возраст:
-2
Пожалуйста, введите положительное число.
Укажите ваш возраст:
30
Вам 30 лет.
```

Программа предлагает ввести возраст до тех пор, пока не будет введено правильное значение. Тем самым гарантируется, что при выходе из цикла `while` переменная `age` будет содержать допустимое значение, которое впоследствии не приведет к сбою программы.

Впрочем, писать код проверки для каждого вызова функции `input()` быстро надоедает. Кроме того, возникает риск пропустить те или иные ошибочные варианты ввода, и в результате в программе окажутся некорректные данные. В этой главе мы рассмотрим использование модуля PyInputPlus для проверки ввода.

Модуль PyInputPlus

Модуль PyInputPlus содержит функции, аналогичные `input()`, которые предназначены для ввода различных типов данных: чисел, дат, адресов электронной почты и т.п. Если пользователь введет недопустимое значение, например неверно отформатированную дату или число за пределами

допустимого диапазона, модуль PyInputPlus предложит ввести данные повторно, как в приведенном выше примере. В модуле PyInputPlus реализованы и другие полезные возможности, такие как ограничение количества повторных запросов или тайм-аут, если пользователь должен уложиться в определенные временны*е* рамки.

Модуль PyInputPlus не является частью стандартной библиотеки Python, поэтому его нужно установить отдельно с помощью утилиты `pip` (сведения по установке сторонних модулей приведены в приложении А). Для этого в командной строке выполните следующую инструкцию:

```
pip install --user pyinputplus
```

Чтобы проверить, корректно ли установлен модуль PyInputPlus, импортируйте его в интерактивной оболочке:

```
>>> import pyinputplus
```

Если при импорте модуля не появилось никаких сообщений об ошибках, значит, он успешно установлен.

Модуль PyInputPlus содержит несколько функций, предназначенных для обработки различных типов вводимых данных.

- **`inputStr()`**. Аналогична встроенной функции `input()`, но поддерживает расширенные возможности модуля PyInputPlus. Ей можно передать пользовательскую функцию для проверки введенных данных.
- **`inputNum()`**. Гарантирует ввод числа и возвращает значение типа `int` или `float`, в зависимости от того, содержит ли введенное число десятичную точку.
- **`inputChoice()`**. Гарантирует выбор одного из предложенных вариантов.
- **`inputMenu()`**. Аналогична функции `inputChoice()`, но отображает меню с числовыми или буквенными вариантами.
- **`inputDatetime()`**. Гарантирует ввод значений даты и времени.
- **`inputYesNo()`**. Гарантирует, что пользователь введет ответ “да/нет”.
- **`inputBool()`**. Аналогична функции `inputYesNo()`, но распознает ответ `'True'` или `'False'` и возвращает соответствующее булево значение.
- **`inputEmail()`**. Гарантирует ввод корректного адреса электронной почты.

- **inputFilepath()**. Гарантирует ввод правильного имени файла (включая путь) и может дополнительно проверять, существует ли файл с таким именем.
- **inputPassword()**. Аналогична встроенной функции input(), но отображает символы * вместо вводимых символов, что позволяет безопасно вводить пароли и другую конфиденциальную информацию.

Эти функции автоматически выводят новый запрос до тех пор, пока не будут введены корректные данные.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum()
пять
'пять' is not a number.
42
>>> response
42
```

Выражение as pyip в инструкции import избавляет от необходимости указывать pyinputplus каждый раз, когда нужно вызвать функцию из этого модуля. Вместо длинного имени используется более короткий псевдоним pyip. В отличие от функции input(), функция inputNum() возвращает значение типа int или float: в данном случае это 42, а не '42'.

Как и в случае функции input(), функциям модуля PyInputPlus можно передать строку приглашения с помощью именованного аргумента prompt.

```
>>> response = input('Введите число: ')
Введите число: 42
>>> response
'42'
>>> import pyinputplus as pyip
>>> response = pyip.inputInt(prompt='Введите число: ')
Введите число: кот
'кот' is not an integer.
Введите число: 42
>>> response
42
```

Чтобы больше узнать о каждой из этих функций, воспользуйтесь функцией help(). Например, если ввести в интерактивной оболочке **help(pyip.inputChoice)**, отобразится справочная информация по функции inputChoice(). Полная документация к модулю доступна по адресу https://pyinputplus.readthedocs.io/.

В отличие от встроенной функции `input()`, функции модуля PyInputPlus имеют ряд дополнительных возможностей проверки, о которых будет рассказано далее.

Именованные аргументы `min`, `max`, `greaterThan` и `lessThan`

Функции `inputNum()`, `inputInt()` и `inputFloat()`, которые работают с целыми числами и числами с плавающей точкой, поддерживают именованные аргументы `min`, `max`, `greaterThan` и `lessThan`, с помощью которых можно задать диапазон допустимых значений. Введите в интерактивной оболочке следующие инструкции.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum('Введите число: ', min=4)
Введите число: 3
Input must be at minimum 4.
Введите число: 4
>>> response
4
>>> response = pyip.inputNum('Введите число: ', greaterThan=4)
Введите число: 4
Input must be greater than 4.
Введите число: 5
>>> response
5
>>> response = pyip.inputNum('>', min=4, lessThan=6)
> 6
Input must be less than 6.
> 3
Input must be at minimum 4.
> 4
>>> response
4
```

Это необязательные аргументы, но если они указаны, то вводимые значения не могут быть меньше аргумента `min` или больше аргумента `max` (но могут быть равны им). Кроме того, вводимые значения должны быть больше, чем аргумент `moreThan`, и меньше, чем аргумент `lessThan` (т.е. они не могут быть равны им).

Именованный аргумент `blank`

По умолчанию ввод пустой строки не допускается, если только для аргумента `blank` не задано значение `True`.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum('Введите число: ')
Введите число: (введено пустое значение)
```

```
Blank values are not allowed.
Введите число: 42
>>> response
42
>>> response = pyip.inputNum(blank=True)
(введено пустое значение)
>>> response
''
```

Используйте аргумент `blank = True` в том случае, когда пользователю разрешается ничего не вводить.

Именованные аргументы `limit`, `timeout` и `default`

По умолчанию функции модуля PyInputPlus будут продолжать запрашивать у пользователя ввод корректных данных бесконечно долго (ну или до тех пор, пока выполняется программа). Если хотите, чтобы функция перестала запрашивать данные после определенного количества попыток или по истечении определенного времени, используйте именованные аргументы `limit` и `timeout`. Целочисленный аргумент `limit` определяет, сколько попыток будет предпринято функцией для получения корректных данных, прежде чем она завершит работу, а целочисленный аргумент `timeout` определяет, сколько секунд отведено пользователю для ввода корректных данных, прежде чем функция завершится.

Если пользователь так и не введет корректных данных за указанное количество попыток или отведенное время, функции сгенерируют исключение `RetryLimitException` или `TimeoutException` соответственно. Например, введите в интерактивной оболочке следующий код.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum(limit=2)
бла-бла-бла
'бла-бла-бла' is not a number.
Enter num: число
'число' is not a number.
Traceback (most recent call last):
    -- Опущено --
pyinputplus.RetryLimitException
>>> response = pyip.inputNum(timeout=10)
42 (введено после 10 секунд ожидания)
Traceback (most recent call last):
    -- Опущено --
pyinputplus.TimeoutException
```

Если помимо этих аргументов указан также аргумент `default`, функция не сгенерирует исключение, а вернет значение, переданное ей с помощью этого аргумента. Введите в интерактивной оболочке следующий код.

```
>>> response = pyip.inputNum(limit=2, default='N/A')
Здравствуй
'Здравствуй' is not a number.
мир
'мир' is not a number.
>>> response
'N/A'
```

Вместо того чтобы генерировать исключение `RetryLimitException`, функция `inputNum()` возвращает строку `'N/A'`.

Именованные аргументы `allowRegexes` и `blockRegexes`

Указывать допустимые значения можно также с помощью регулярных выражений. Именованные аргументы `allowRegexes` и `blockRegexes` позволяют задать списки регулярных выражений, определяющих, какие значения функция принимает в качестве допустимых, а какие — отклоняет. Например, введите в интерактивной оболочке следующий код, чтобы функция `inputNum()` помимо обычных чисел поддерживала римские цифры.

```
>>> import pyinputplus as pyip
>>> response=pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+',
                           r'zero'])
XLII
>>> response
'XLII'
>>> response=pyip.inputNum(allowRegexes=[r'(i|v|x|l|c|d|m)+',
                           r'zero'])
xlii
>>> response
'xlii'
```

Конечно, эти регулярные выражения задают только буквы, которые разрешается вводить пользователю. Порядок цифр в числе может оказаться неправильным, например `'XVX'` или `'MILLI'`, потому что регулярное выражение `r'(I|V|X|L|C|D|M)+'` допускает такое число.

С помощью именованного аргумента `blockRegexes` можно также указать список регулярных выражений, которые функция не будет принимать. Введите в интерактивной оболочке следующий код, чтобы функция `inputNum()` не принимала четные числа.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputNum(blockRegexes=[r'[02468]$'])
42
This response is invalid.
44
This response is invalid.
```

```
43
>>> response
43
```

Если указать оба аргумента — и `allowRegexes`, и `blockRegexes`, — то список разрешений перекрывает список блокировок. Например, введите в интерактивной оболочке следующий код, который разрешает ввод слов `'caterpillar'` и `'category'`, но блокирует любые другие строки, содержащие слово `'cat'`.

```
>>> import pyinputplus as pyip
>>> response = pyip.inputStr(allowRegexes=[r'caterpillar',
...                                        'category'],
...                           blockRegexes=[r'cat'])
cat
This response is invalid.
catastrophe
This response is invalid.
category
>>> response
'category'
```

Функции модуля PyInputPlus помогут избавить вас от утомительного написания кода проверки вводимых данных. Полная документация к этому модулю доступна по адресу `https://pyinputplus.readthedocs.io/`.

Передача пользовательской функции проверки в функцию `inputCustom()`

Можно написать функцию, реализующую требуемую логику проверки, и передать ее функции `inputCustom()`. Допустим, необходимо, чтобы пользователь ввел последовательность цифр, в сумме равных 10. Функции `pyinputplus.inputAddsUpToTen()` не существует, но можно создать свою собственную функцию, которая:

- имеет один строковый аргумент (значение, введенное пользователем);
- генерирует исключение, если строка не проходит проверку;
- возвращает `None` (инструкция `return` может быть опущена), если функция `inputCustom()` должна вернуть строку без изменений;
- возвращает значение, отличное от `None`, если функция `inputCustom()` должна вернуть строку, отличную от той, которую ввел пользователь;
- передается в качестве первого аргумента функции `inputCustom()`.

Например, можно создать пользовательскую функцию `addUpToTen()` и передать ее функции `inputCustom()`. При этом вызов должен выглядеть как `inputCustom(addUpToTen)`, а не `inputCustom(addUpToTen())`, потому что мы передаем ссылку на функцию `addUpToTen()`, а не возвращаемое ею значение.

```
>>> import pyinputplus as pyip
>>> def addsUpToTen(numbers):
...     numbersList = list(numbers)
...     for i, digit in enumerate(numbersList):
...         numbersList[i] = int(digit)
...     if sum(numbersList) != 10:
...         raise Exception('Сумма должна быть 10, а не %s.' %
(sum(numbersList)))
...     return int(numbers)    # вернуть целое число
...
>>> response = pyip.inputCustom(addsUpToTen) # без скобок после имени
123
Сумма должна быть 10, а не 6.
1235
Сумма должна быть 10, а не 11.
1234
>>> response # функция inputCustom() возвращает целое число, а не строку
1234
>>> response = pyip.inputCustom(addsUpToTen)
Здравствуй
invalid literal for int() with base 10: 'З'
55
>>> response
55
```

Функция `inputCustom()` тоже поддерживает именованные аргументы `blank`, `limit`, `timeout`, `default`, `allowRegexes` и `blockRegexes`. Написание собственной функции проверки целесообразно в том случае, когда трудно или невозможно написать регулярное выражение для проверки допустимых данных, как в приведенном выше примере.

Проект: как занять дурака на несколько часов

Воспользуемся модулем PyInputPlus для создания простой программы, которая выполняет следующие действия:

1) спрашивает пользователя, не хотел бы он узнать, как занять дурака на протяжении нескольких часов;
2) завершает работу, если пользователь отвечает “нет”;
3) возвращается к п. 1, если пользователь отвечает “да”.

Поскольку мы не знаем, будет ли пользователь вводить что-то кроме 'да' или 'нет', необходимо проверить правильность введенных данных. Было бы также удобно, чтобы пользователь мог вводить 'д' или 'н' вместо полных слов. Эти проверки выполняет функция inputYesNo() из модуля PyInputPlus, которая независимо от регистра введенных символов возвращает строку 'да' или 'нет' в нижнем регистре.

Результаты работы программы будут выглядеть примерно так.

```
Хотите узнать, как занять дурака на несколько часов?
конечно
'конечно' is not a valid yes/no response.
Хотите узнать, как занять дурака на несколько часов?
да
Хотите узнать, как занять дурака на несколько часов?
д
Хотите узнать, как занять дурака на несколько часов?
Да
Хотите узнать, как занять дурака на несколько часов?
ДА
Хотите узнать, как занять дурака на несколько часов?
ДА!!!!!!
'ДА!!!!!!' is not a valid yes/no response.
Хотите узнать, как занять дурака на несколько часов?
СКАЖИ МНЕ, КАК.
'СКАЖИ МНЕ, КАК.' is not a valid yes/no response.
Хотите узнать, как занять дурака на несколько часов?
нет
Спасибо! Желаю хорошего дня.
```

Откройте новую вкладку в редакторе файлов и сохраните файл под именем *idiot.py*. Затем введите следующий код:

```
import pyinputplus as pyip
```

В результате будет импортирован модуль PyInputPlus. Поскольку имя pyinputplus слишком длинное, мы назначаем ему псевдоним pyip.

```
while True:
    prompt = 'Хотите узнать, как занять дурака на несколько часов?'
    response = pyip.inputYesNo(prompt)
```

Выражение while True: создает бесконечный цикл, который выполняется до тех пор, пока не встретится инструкция break. В цикле вызывается функция pyip.inputYesNo(), которая не завершится до тех пор, пока пользователь не введет корректный ответ.

```
    if response == 'no':
        break
```

Функция `pyip.inputYesNo()` гарантированно возвращает `'yes'` или `'no'`. Если возвращается `'no'`, программа выходит из бесконечного цикла и продолжает выполняться до последней строки, после чего прощается с пользователем:

```
print('Спасибо! Желаю хорошего дня.')
```

В противном случае цикл повторяется снова.

Функция `inputYesNo()` поддерживает языки, отличные от английского, благодаря именованным аргументам `yesVal` и `noVal`. Например, русскоязычная версия программы должна содержать следующий код.

```
    response = pyip.inputYesNo(prompt, yesVal= 'да', noVal='нет')
    if response == 'нет':
```

Теперь пользователь может ввести `'да'` или `'д'` (в нижнем или верхнем регистре) вместо `'yes'` или `'y'` в качестве утвердительного ответа.

Проект: тест на умножение

Модуль PyInputPlus может оказаться полезным для создания теста на умножение с лимитом времени. Благодаря именованным аргументам `allowRegexes`, `blockRegexes`, `timeout` и `limit` функции `pyip.inputStr()` бо́льшая часть операций реализуется в самом модуле PyInputPlus. Чем меньше кода предстоит написать, тем быстрее можно получить готовую программу. Мы напишем программу, которая выводит пользователю 10 заданий на умножение. Откройте в редакторе файлов новую вкладку и сохраните файл под именем *multiplicationQuiz.py*.

Сначала необходимо импортировать модули `pyinputplus`, `random` и `time`. Мы будем отслеживать, сколько вопросов задала программа и сколько правильных ответов дал пользователь, с помощью переменных `numberOfQuestions` и `correctAnswers`. В цикле `for` будет случайным образом 10 раз выдаваться задание на умножение.

```
import pyinputplus as pyip
import random, time

numberOfQuestions = 10
correctAnswers = 0
for questionNumber in range(numberOfQuestions):
```

В цикле `for` программа выбирает две перемножаемые цифры. Эти цифры отображаются в приглашении `#Q: N × N=`, где `Q` — номер вопроса (от 1 до 10), а `N` — числа, которые нужно умножить.

```
# Выбираем два случайных числа
num1 = random.randint(0, 9)
num2 = random.randint(0, 9)
prompt = '#%s: %s x %s = ' % (questionNumber, num1, num2)
```

Основная логика программы реализована в функции `pyip.inputStr()`. Именованный аргумент `allowRegexes` представляет собой список, содержащий регулярное выражение `'^%s$'`, где `%s` заменяется правильным ответом. Символы `^` и `%` гарантируют, что ответ начинается и заканчивается правильным числом, хотя функции PyInputPlus сначала удаляют любые ведущие и замыкающие пробелы на тот случай, если пользователь случайно нажмет пробела до или после ответа. Именованный аргумент `blocklistRegexes` содержит список с одним кортежем `('.*', 'Неправильно!')`. Первая строка в кортеже — это регулярное выражение, которое соответствует любой возможной строке. Следовательно, если пользователь вводит неправильный ответ, программа отклоняет его. В таком случае отображается строка `'Неправильно!'`, после чего пользователю предлагается ответить снова. Кроме того, аргументы `timeout=8` и `limit=3` гарантируют, что у пользователя есть только 8 секунд и 3 попытки дать правильный ответ.

```
try:
    # Правильные ответы задаются аргументом allowRegexes;
    # неправильные ответы задаются аргументом blockRegexes
    # (в случае неправильного ответа отображается
    # пользовательское сообщение)
    pyip.inputStr(prompt, allowRegexes=['^%s$' % (num1 * num2)], \
                  blockRegexes=[('.*', 'Неправильно!')], \
                  timeout=8, limit=3)
```

Если пользователь отвечает по истечении 8-секундного тайм-аута, то даже в случае правильного ответа функция `pyip.inputStr()` генерирует исключение `TimeoutException`. Если пользователь отвечает неправильно более трех раз, генерируется исключение `RetryLimitException`. Оба этих типа исключений определены в модуле PyInputPlus, поэтому их нужно предварять префиксом `pyip`.

```
except pyip.TimeoutException:
    print('Время истекло!')
except pyip.RetryLimitException:
    print('Закончилось количество попыток!')
```

Помните, что блок `else` может следовать не только за блоком `if` или `elif`, но и за блоком `except`. Следующий код будет выполняться, если в блоке `try` не было сгенерировано исключение. В нашем случае это означает, что пользователь ввел правильный ответ.

```
    else:
        # Этот блок выполняется, если в блоке try
        # не возникло исключений
        print('Правильно!')
        correctAnswers += 1
```

Независимо от того, какое из трех сообщений (`'Время истекло!'`, `'Закончилось количество попыток!'` или `'Правильно'`) отображается, в конце цикла делается секундная пауза, которая дает пользователю время на прочтение сообщения. После того как программа задала 10 вопросов, пользователь увидит количество правильных ответов.

```
    time.sleep(1)    # короткая пауза, позволяющая пользователю
                     # увидеть результат
print('Счет: %s/%s'%(correctAnswers, numberOfQuestions))
```

Модуль PyInputPlus достаточно гибкий, благодаря чему его можно использовать в самых разных программах, которые принимают ввод пользователя с клавиатуры.

Резюме

Многие программисты забывают писать код для проверки вводимых данных, но без него в программах практически наверняка будут возникать ошибки. Значения, которые вы ожидаете от пользователей, и значения, которые они фактически вводят, могут оказаться совершенно разными, и программы должны быть достаточно надежными, чтобы справляться с такими ситуациями. Для создания кода проверки вводимых данных можно использовать регулярные выражения, но обычно проще использовать готовый модуль, такой как PyInputPlus. Импортируйте этот модуль с помощью инструкции `import pyinputplus as pyip`, чтобы при вызове функций модуля указывать более короткий псевдоним `pyip`.

Модуль PyInputPlus содержит функции для ввода различных типов данных, включая строки, числа, даты, булевы значения, варианты “да/нет”, адреса электронной почты и файлы. В то время как функция `input()` всегда возвращает строку, функции модуля PyInputPlus возвращают значения соответствующего типа данных. Функция `inputChoice()` позволяет выбрать один из нескольких предварительно заданных вариантов, а функция `inputMenu()` добавляет цифры или буквы к вариантам выбора.

Все эти функции поддерживают следующие стандартные возможности: удаление ведущих и замыкающих пробелов, установка тайм-аута и количества допустимых повторов с помощью именованных аргументов `timeout` и `limit`, а также передача списков регулярных выражений с помощью аргументов `allowRegexes` и `blockRegexes`, позволяющих принимать или отвергать определенные ответы. Вам больше не нужно писать утомительные циклы `while`, в которых проверяется правильность введенных данных и выводятся повторные запросы.

Если ни одна из функций модуля PyInputPlus не соответствует вашим задачам, можно написать собственную функцию проверки и передать ее функции `inputCustom()`. Полный список функций модуля доступен по адресу `https://pyinputplus.readthedocs.io/en/latest/`. В онлайн-документации содержится гораздо больше информации, чем было приведено в этой главе. Не стоит изобретать велосипед — лучше научиться использовать этот модуль, чем писать (и отлаживать!) собственный код.

Теперь, когда вы умеете работать с текстом и проверять его правильность, пора научиться считывать и записывать файлы, хранящиеся на жестком диске. Этой теме посвящена следующая глава.

Контрольные вопросы

1. Входит ли модуль PyInputPlus в стандартную библиотеку Python?
2. Почему модуль PyInputPlus обычно импортируют с помощью инструкции `import pyinputplus as pyip`?
3. Чем отличается функция `inputInt()` от функции `inputFloat()`?
4. Как с помощью модуля PyInputPlus гарантировать, что пользователь введет целое число в диапазоне от `0` до `99`?
5. Что передается с помощью именованных аргументов `allowRegexes` и `blockRegexes`?
6. Что сделает функция `inputStr(limit=3)`, если три раза ввести пустую строку?
7. Что сделает функция `inputStr(limit=3, default='привет')`, если три раза ввести пустую строку?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Изготовитель бутербродов

Напишите программу, которая спрашивает пользователя о его бутербродных предпочтениях. Программа должна использовать модуль PyInputPlus, чтобы гарантировать ввод корректных данных.

- Используйте функцию `inputMenu()` для определения типа хлеба: цельнозерновой, белый или ржаной.
- Используйте функцию `inputMenu()` для определения типа белкового продукта: курица, индейка, ветчина или тофу.
- Используйте функцию `inputYesNo()`, чтобы спросить, хочет ли пользователь добавить сыр.
- Если пользователь ответил утвердительно, используйте функцию `inputMenu()`, чтобы узнать тип сыра: чеддер, швейцарский или моцарелла.
- Используйте функцию `inputYesNo()`, чтобы узнать у пользователя, хочет ли он добавить майонез, горчицу, салат или помидор.
- Используйте функцию `inputInt()`, чтобы узнать, сколько бутербродов хочет пользователь. Убедитесь в том, что это число не меньше 1.

Придумайте цены для каждого из параметров бутерброда, и пусть программа отобразит общую стоимость после того, как пользователь сделает свой выбор.

Собственный тест на умножение

Чтобы оценить реальные возможности модуля PyInputPlus, попробуйте воссоздать тест на умножение без использования этого модуля. Программа должна предложить пользователю 10 заданий на умножение в диапазоне от 0·0 до 9·9. Необходимо реализовать следующий функционал.

- Если пользователь вводит правильный ответ, программа в течение одной секунды отображает сообщение “Правильно!” и переходит к следующему вопросу.
- Пользователь получает три попытки для ввода правильного ответа, прежде чем программа перейдет к следующему вопросу.
- Через восемь секунд после первого отображения вопроса он помечается как неправильный, даже если пользователь вводит правильный ответ после 8-секундной паузы.

Сравните свой код с кодом на основе модуля PyInputPlus, который был приведен в главе.