Часть II

РЕШЕНИЯ

Глава 9

Введение в программирование

Упражнение 1. Почтовый адрес

```
##
# Отобразить почтовый адрес пользователя
#
print("Бен Стивенсон")
print("Департамент теории вычислительных систем")
print("Университет Калгари")
print("2500 Университетское шоссе NW")
print("Калгари, Альберта T2N 1N4")
print("Канада")
```

Упражнение 3. Площадь комнаты

```
##
# Вычислить площадь комнаты
#

# Считываем ввод пользователя
length = float(input("Введите длину комнаты (м): "))
width = float(input("Введите ширину комнаты (м): "))

# Вычислим площадь комнаты
area = length * width

# Отобразим результат
print("Площадь комнаты равна", area, "кв.м.")
```

Функция `float` используется для преобразования пользовательского ввода в нужный тип данных.

В Python умножение выполняется при помощи оператора *.

Упражнение 4. Площадь садового участка

```
##
# Вычисляем площадь садового участка в акрах
#
```

```
SQFT_PER_ACRE = 43560

# Запрашиваем информацию у пользователя
length = float(input("Введите длину участка (футы): "))
width = float(input("Введите ширину участка (футы): "))

# Вычислим площадь в акрах
acres = length * width / SQFT_PER_ACRE

# Отобразим результат
print("Площадь садового участка равна", acres, "акров")
```

Упражнение 5. Сдаем бутылки

```
##
# Вычисляем доход от сданной тары
#
LESS_DEPOSIT = 0.10
MORE_DEPOSIT = 0.25

# Запрашиваем у пользователя количество бутылок каждого вида
less = int(input("Сколько у вас бутылок объемом 1 литр и меньше? "))
more = int(input("Сколько у вас бутылок объемом больше 1 литра? "))

# Вычисляем сумму
refund = less * LESS_DEPOSIT + more * MORE_DEPOSIT

# Отобразим результат
print("Ваша выручка составит $%.2f." % refund)
```

> Спецификатор формата %.2f указывает Python на то, что необходимо форматировать значение в виде числа с плавающей запятой с двумя десятичными знаками.

Упражнение 6. Налоги и чаевые

```
##
# Вычисляем налог и сумму чаевых в ресторане
#
TAX_RATE = 0.05
TIP_RATE = 0.18
```

> В моем регионе налог составляет 5 %. В языке Python 5 % и 18 % представляются как 0.05 и 0.18 соответственно.

```
# Запрашиваем сумму счета у пользователя
cost = float(input("Введите сумму счета: "))
```

```
# Вычисляем сумму налога и чаевых
tax = cost * TAX_RATE
tip = cost * TIP_RATE
total = cost + tax + tip

# Отобразим результат
print("Налог составил %.2f, чаевые - %.2f, общая сумма", \
        "заказа: %.2f" % (tax, tip, total))
```

Знак обратной косой черты (\) называется символом продолжения строки. Он сообщает Python о том, что инструкция будет продолжена на следующей строке. Не вводите пробелы и иные символы, включая символ табуляции, после косой черты.

Упражнение 7. Сумма первых n положительных чисел

```
##
# Рассчитываем сумму первых n положительных чисел
#
# Запрашиваем значение числа n у пользователя
n = int(input("Введите положительное число: "))

# Рассчитываем сумму
sm = n * (n + 1) / 2
```

В Python есть встроенная функция с именем `sum`, поэтому для нашей переменной мы выбрали другое имя.

```
# Отобразим результат
print("Сумма первых", n, "положительных чисел равна", sm)
```

Упражнение 10. Арифметика

```
##
# Демонстрируем математические операции и использование модуля math
#
from math import log10
```

Необходимо импортировать функцию `log10` из модуля `math`, прежде чем она будет использована. Обычно все инструкции импорта располагаются в начале кода.

```
# Запрашиваем два целых числа у пользователя
a = int(input("Введите число a: "))
```

```
b = int(input("Введите число b: "))

# Рассчитываем и отображаем сумму, разницу, произведение, частное и остаток от деления
print(a, "+", b, "=", a + b)
print(a, "-", b, "=", a - b)
print(a, "*", b, "=", a * b)
print(a, "/", b, "=", a / b)
print(a, "%", b, "=", a % b)
```

Остаток от деления двух чисел вычисляется при помощи функции %, для возведения в степень есть оператор **.

```
# Рассчитываем десятичный логарифм и степень
print("Десятичный логарифм числа", a, "равен", log10(a))
print(a, "в степени", b, "равно", a ** b)
```

Упражнение 13. Размен

```
##
# Рассчитываем минимальное количество монет для представления указанной суммы
#
CENTS_PER_TOONIE = 200
CENTS_PER_LOONIE = 100
CENTS_PER_QUARTER = 25
CENTS_PER_DIME =10
CENTS_PER_NICKEL =5

# Запрашиваем у пользователя сумму в центах
cents = int(input("Введите сумму в центах: "))

# Определим количество двухдолларовых монет путем деления суммы на 200. Затем вычислим
# оставшуюся сумму для размена, рассчитав остаток от деления
print(" ", cents // CENTS_PER_TOONIE, "двухдолларовых монет")
cents = cents % CENTS_PER_TOONIE
```

Деление без остатка в Python выполняется при помощи оператора //. При этом результат всегда будет округлен в нижнюю сторону, что нам и требуется.

```
# Повторяем эти действия для остальных монет
print(" ", cents // CENTS_PER_LOONIE, "однодолларовых монет")
cents = cents % CENTS_PER_LOONIE
print(" ", cents // CENTS_PER_QUARTER, "25-центовых монет")
cents = cents % CENTS_PER_QUARTER
print(" ", cents // CENTS_PER_DIME, "10-центовых монет")
cents = cents % CENTS_PER_DIME
print(" ", cents // CENTS_PER_NICKEL, "5-центовых монет")
cents = cents % CENTS_PER_NICKEL

# Отобразим остаток в центах
print(" ", cents, "центов")
```

Упражнение 14. Рост

```
##
# Преобразуем рост в футах и дюймах в сантиметры
#
IN_PER_FT = 12
CM_PER_IN = 2.54

# Запрашиваем рост у пользователя
print("Введите рост:")
feet = int(input("Количество футов: "))
inches = int(input("Количество дюймов: "))

# Переводим в сантиметры
cm = (feet * IN_PER_FT + inches) * CM_PER_IN

# Отобразим результат
print("Ваш рост в сантиметрах:", cm)
```

Упражнение 17. Теплоемкость

```
##
# Вычислить количество энергии, требуемое для нагрева воды, а также стоимость нагрева
#
# Определим константы для удельной теплоемкости воды и стоимости электричества
WATER_HEAT_CAPACITY = 4.186
ELECTRICITY_PRICE = 8.9
J_TO_KWH = 2.777e-7
```

> Python позволяет записывать числа в научной нотации, когда коэффициент располагается слева от буквы e, а порядок – справа. Таким образом, число $2{,}777*10^{-7}$ может быть записано как 2,777e–7.

```
# Запрашиваем у пользователя объем воды и требуемое изменение температуры
volume = float(input("Объем воды в миллилитрах: "))
d_temp = float(input("Повышение температуры (в градусах Цельсия): "))
```

> Поскольку вода обладает плотностью 1 грамм на миллилитр, в данном упражнении можно взаимозаменять граммы и миллилитры. Решение запрашивать у пользователя объем жидкости в миллилитрах было принято из соображений удобства, поскольку люди привыкли представлять себе объем кофейной кружки, а не массу воды в ней.

```
# Вычисляем количество энергии в джоулях
q = volume * d_temp * WATER_HEAT_CAPACITY
```

```
# Отображаем результат в джоулях
print("Потребуется %d Дж энергии." % q)

# Вычисляем стоимость
kwh = q * J_TO_KWH
cost = kwh * ELECTRICITY_PRICE

# Отображаем стоимость
print("Стоимость энергии: %.2f центов." % cost)
```

Упражнение 19. Свободное падение

```
##
# Рассчитываем скорость объекта, отпущенного с определенной высоты,
# в момент столкновения с землей
#
from math import sqrt

# Определяем константу ускорения свободного падения
GRAVITY = 9.8

# Запрашиваем высоту, с которой объект был отпущен
d = float(input("Высота отпускания объекта (в метрах): "))

# Рассчитываем финальную скорость
vf = sqrt(2 * GRAVITY * d)
```

v_i^2 не была включена в формулу расчета v_f, поскольку $v_i = 0$.

```
# Отобразим результат
print("Объект достигнет земли на скорости %.2f м/с." % vf)
```

Упражнение 23. Площадь правильного многоугольника

```
##
# Вычисляем площадь правильного многоугольника
#
from math import tan, pi

# Запрашиваем информацию у пользователя
s = float(input("Введите длину сторон: "))
n = int(input("Введите число сторон: "))
```

Сразу конвертируем *n* в целочисленное значение, а не в число с плавающей запятой, поскольку у многоугольника не может быть дробного количества сторон.

```
# Вычисляем площадь многоугольника
area = (n * s ** 2) / (4* tan(pi / n))

# Отобразим результат
print("Площадь многоугольника равна", area)
```

Упражнение 25. Единицы времени (снова)

```
##
# Переводим секунды в дни, часы, минуты и секунды
#
SECONDS_PER_DAY = 86400
SECONDS_PER_HOUR = 3600
SECONDS_PER_MINUTE = 60

# Запрашиваем у пользователя длительность в секундах
seconds = int(input("Введите количество секунд: "))

# Переводим введенное значение в дни, часы, минуты и секунды
days = seconds / SECONDS_PER_DAY
seconds = seconds % SECONDS_PER_DAY
hours = seconds / SECONDS_PER_HOUR
seconds = seconds % SECONDS_PER_HOUR
minutes = seconds / SECONDS_PER_MINUTE
seconds = seconds % SECONDS_PER_MINUTE

# Отобразим результат в требуемом формате
print("Длительность:", \
        "%d:%02d:%02d:%02d." % (days, hours, minutes, seconds))
```

Спецификатор формата `%02d` указывает Python на то, что необходимо форматировать целочисленное значение в виде двух цифр путем добавления ведущего нуля при необходимости.

Упражнение 29. Температура с учетом ветра

```
##
# Вычисляем коэффициент охлаждения ветром
#
WC_OFFSET = 13.12
WC_FACTOR1 = 0.6215
WC_FACTOR2 = -11.37
WC_FACTOR3 = 0.3965
WC_EXPONENT = 0.16

# Запрашиваем у пользователя температуру воздуха и скорость ветра
temp = float(input("Температура воздуха (градусы Цельсия): "))
```

Вычисление коэффициента охлаждения ветром потребовало ввода нескольких констант, которые были определены учеными и медиками.

```
speed = float(input("Скорость ветра (км/ч): "))

# Определяем коэффициент охлаждения ветром
wci = WC_OFFSET + \
WC_FACTOR1 * temp + \
WC_FACTOR2 * speed ** WC_EXPONENT + \
WC_FACTOR3 * temp * speed ** WC_EXPONENT

# Отобразим результат, округленный до ближайшего целого
print("Коэффициент охлаждения ветром равен", round(wci))
```

Упражнение 33. Сортировка трех чисел

```
##
# Сортируем три числа по возрастанию
#
# Запрашиваем числа у пользователя и записываем их в переменные a, b и c
a = int(input("Введите первое число: "))
b = int(input("Введите второе число: "))
c = int(input("Введите третье число: "))
mn = min(a, b, c) # Минимальное значение
mx = max(a, b, c) # Максимальное значение
md = a + b + c - mn - mx # Среднее значение

# Отобразим результат
print("Числа в порядке возрастания:")
print(" ", mn)
print(" ", md)
print(" ", mx)
```

Поскольку слова `min` и `max` являются в Python зарезервированными, мы не можем использовать их для именования переменных. Вместо этого мы будем хранить значения минимума и максимума в переменных с именами `mn` и `mx`.

Упражнение 34. Вчерашний хлеб

```
##
# Вычисляем стоимость вчерашнего хлеба
#
BREAD_PRICE = 3.49
DISCOUNT_RATE = 0.60

# Запрашиваем данные у пользователя
num_loaves = int(input("Введите количество вчерашних буханок хлеба: "))

# Вычисляем скидку и общую стоимость
regular_price = num_loaves * BREAD_PRICE
discount = regular_price * DISCOUNT_RATE
total = regular_price - discount

# Отобразим результат в нужном формате
print("Номинальная цена: %5.2f" % regular_price)
```

```
print("Сумма скидки: %5.2f" % discount)
print("Итого: %5.2f" % total)
```

Формат %5.2f предполагает использование пяти знакомест для отображения чисел, при этом под десятичные знаки должно быть отведено два места. Это поможет внешне выровнять столбцы в таблице при разном количестве цифр в значениях.

Глава 10

Принятие решений

Упражнение 35. Чет или нечет?

```
##
# Определяем и выводим на экран информацию о том, четное введенное число или нечетное
#
# Запрашиваем целое число у пользователя
num = int(input("Введите целое число: "))

# Используем оператор взятия остатка от деления
if num % 2 == 1:
    print(num, "нечетное.")
else:
    print(num, "четное.")
```

Остаток от деления четного числа на 2 всегда будет давать 0, а нечетного – 1.

Упражнение 37. Гласные и согласные

```
##
# Определяем, является буква гласной или согласной
#
# Запрашиваем ввод буквы с клавиатуры
letter = input("Введите букву латинского алфавита: ")

# Классифицируем букву и выводим результат
if letter == "a" or letter == "e" or \
        letter == "i" or letter == "o" or \
        letter == "u":
    print("Это гласная буква.")
elif letter == "y":
    print("Иногда буква гласная, иногда согласная.")
else:
    print("Это согласная буква.")
```

Эта версия программы работает только для ввода букв в нижнем регистре. Вы можете добавить проверку на заглавные буквы, если есть такая необходимость.

Упражнение 38. Угадайте фигуру

```
##
# Определяем вид фигуры по количеству сторон
```

```
#
# Запрашиваем у пользователя количество сторон
nsides = int(input("Введите количество сторон фигуры: "))

# Определяем вид фигуры, оставляя его пустым, если введено некорректное число
name = ""
if nsides == 3:
    name = "треугольник"
elif nsides == 4:
    name = "прямоугольник"
elif nsides == 5:
    name = "пятиугольник"
elif nsides == 6:
    name = "шестиугольник"
elif nsides == 7:
    name = "семиугольник"
elif nsides == 8:
    name = "восьмиугольник"
elif nsides == 9:
    name = "девятиугольник"
elif nsides == 10:
    name = "десятиугольник"

# Выводим ошибку ввода
if name == "":
    print("Введенное количество сторон не поддерживается программой.")
else:
    print("Эта фигура:", name)
```

Пустую строку мы используем как индикатор. Если введенное пользователем значение окажется за границами обрабатываемого нами диапазона, значение переменной `name` останется пустым, что позволит нам позже вывести сообщение об ошибке.

Упражнение 39. Сколько дней в месяце?

```
##
# Определяем количество дней в месяце
#
# Запрашиваем у пользователя название месяца
month = input("Введите название месяца: ")

# Вычисляем количество дней в месяце
days = 31
```

Изначально считаем, что в месяце 31 день, после чего постепенно корректируем это предположение.

```
if month == "Апрель" or month == "Июнь" or \
        month == "Сентябрь" or month == "Ноябрь":
    days = 30
```

```
elif month == "Февраль":
    days = "28 или 29"
```

Для февраля присваиваем переменной days строковое значение. Это позволит нам вывести информацию о том, что в этом месяце количество дней может варьироваться.

```
# Выводим результат
print("Количество дней в месяце", month, "равно", days)
```

Упражнение 41. Классификация треугольников

```
##
# Классифицируем треугольники на основании длин их сторон
#
# Запрашиваем у пользователя длины сторон треугольника
side1 = float(input("Введите длину первой стороны: "))
side2 = float(input("Введите длину второй стороны: "))
side3 = float(input("Введите длину третьей стороны: "))

# Определяем вид треугольника
if side1 == side2 and side2 == side3:
    tri_type = "равносторонний"
elif side1 == side2 or side2 == side3 or \
        side3 == side1:
    tri_type = "равнобедренный"
else:
    tri_type = "разносторонний"

# Отображаем вид треугольника
print("Это", tri_type, "треугольник")
```

При проверке того, является ли треугольник равносторонним, можно было добавить условие равенства side1 и side3, но в этом нет необходимости, поскольку оператор равенства (==) является транзитивным.

Упражнение 42. Узнать частоту по ноте

```
##
# Преобразуем название ноты в частоту
#
C4_FREQ = 261.63
D4_FREQ = 293.66
E4_FREQ = 329.63
F4_FREQ = 349.23
G4_FREQ = 392.00
A4_FREQ = 440.00
B4_FREQ = 493.88

# Запрашиваем у пользователя название ноты
name = input("Введите название ноты в виде буквы и цифры, например C4: ")
```

```
# Сохраняем название ноты и номер октавы в разных переменных
note = name[0]
octave = int(name[1])

# Получаем частоту ноты четвертой октавы
if note == "C":
    freq = C4_FREQ
elif note == "D":
    freq = D4_FREQ
elif note == "E":
    freq = E4_FREQ
elif note == "F":
    freq = F4_FREQ
elif note == "G":
    freq = G4_FREQ
elif note == "A":
    freq = A4_FREQ
elif note == "B":
    freq = B4_FREQ

# Адаптируем частоту к конкретной октаве
freq = freq / 2 ** (4 - octave)

# Выводим результат
print("Частота ноты", name, "равна", freq)
```

Упражнение 43. Узнать ноту по частоте

```
##
# Запрашиваем у пользователя частоту ноты и определяем ее название
#
C4_FREQ = 261.63
D4_FREQ = 293.66
E4_FREQ = 329.63
F4_FREQ = 349.23
G4_FREQ = 392.00
A4_FREQ = 440.00
B4_FREQ = 493.88
LIMIT = 1

# Запрашиваем у пользователя частоту ноты
freq = float(input("Введите частоту ноты (Гц): "))

# Определяем ноту по частоте. Если нота не найдена, присваиваем переменной
# пустую строку
if freq >= C4_FREQ - LIMIT and freq <= C4_FREQ + LIMIT:
    note = "C4"
elif freq >= D4_FREQ - LIMIT and freq <= D4_FREQ + LIMIT:
    note = "D4"
```

```
elif freq >= E4_FREQ - LIMIT and freq <= E4_FREQ + LIMIT:
    note = "E4"
elif freq >= F4_FREQ - LIMIT and freq <= F4_FREQ + LIMIT:
    note = "F4"
elif freq >= G4_FREQ - LIMIT and freq <= G4_FREQ + LIMIT:
    note = "G4"
elif freq >= A4_FREQ - LIMIT and freq <= A4_FREQ + LIMIT:
    note = "A4"
elif freq >= B4_FREQ - LIMIT and freq <= B4_FREQ + LIMIT:
    note = "B4"
else:
    note = ""

# Отображаем результат или сообщение об ошибке
if note == "":
    print("Ноты с заданной частотой не существует.")
else:
    print("Введенная частота примерно соответствует ноте", note)
```

Упражнение 47. Определение времени года

```
##
# Определяем и выводим название сезона по дате
#
# Запрашиваем у пользователя дату
month = input("Введите название месяца: ")
day = int(input("Введите день: "))
```

> Представленное решение содержит множество блоков `elif`, чтобы максимально упростить сценарий. Можно уменьшить количество блоков `elif` за счет усложнения общей условной конструкции.

```
# Определяем сезон
if month == "Январь" or month == "Февраль":
    season = "Зима"
elif month == "Март":
    if day < 20:
        season = "Зима"
    else:
        season = "Весна"
elif month == "Апрель" or month == "Май":
    season = "Весна"
elif month == "Июнь":
    if day < 21:
        season = "Весна"
    else:
```

```
        season = "Лето"
elif month == "Июль" or month == "Август":
    season = "Лето"
elif month == "Сентябрь":
    if day < 22:
        season = "Лето"
    else:
        season = "Осень"
elif month == "Октябрь" or month == "Ноябрь":
    season = "Осень"
elif month == "Декабрь":
    if day < 21:
        season = "Осень"
    else:
        season = "Зима"

# Выводим результат
print(month, day, "соответствует сезону", season)
```

Упражнение 49. Китайский гороскоп

```
##
# Определяем животное, ассоциированное с введенным годом в китайском гороскопе
#
# Запрашиваем у пользователя год
year = int(input("Введите год: "))

# Определяем ассоциированное с годом животное
if year % 12 == 8:
    animal = "Дракон"
elif year % 12 == 9:
    animal = "Змея"
elif year % 12 == 10:
    animal = "Лошадь"
elif year % 12 == 11:
    animal = "Коза"
elif year % 12 == 0:
    animal = "Обезьяна"
elif year % 12 == 1:
    animal = "Петух"
elif year % 12 == 2:
    animal = "Собака"
elif year % 12 == 3:
    animal = "Свинья"
elif year % 12 == 4:
    animal = "Крыса"
elif year % 12 == 5:
    animal = "Бык"
```

```
elif year % 12 == 6:
    animal = "Тигр"
elif year % 12 == 7:
    animal = "Кролик"

# Выводим результат
print("Год %d ассоциирован с животным: %s." % (year, animal))
```

При форматировании нескольких элементов в одной строке они перечисляются через запятую в круглых скобках справа от оператора %.

Упражнение 52. Буквенные оценки – в числовые

```
##
# Преобразуем буквенные оценки в числовые
#
A = 4.0
A_MINUS = 3.7
B_PLUS = 3.3
B = 3.0
B_MINUS = 2.7
C_PLUS = 2.3
C = 2.0
C_MINUS = 1.7
D_PLUS = 1.3
D = 1.0
F = 0
INVALID = -1

# Запрашиваем буквенную оценку у пользователя
letter = input("Введите буквенную оценку: ")
letter = letter.upper()
```

Инструкция `letter = letter.upper()` переводит все символы нижнего регистра в верхний, сохраняя результат в первоначальную переменную. Это позволит работать с оценками, введенными в нижнем регистре, без включения в программу дополнительных условных конструкций.

```
# Преобразуем оценку из буквенной в числовую, используя значение -1 как индикатор,
# означающий ошибочный ввод
if letter == "A+" or letter == "A":
    gp = A
elif letter == "A-":
    gp = A_MINUS
```

```
elif letter == "B+":
    gp = B_PLUS
elif letter == "B":
    gp = B
elif letter == "B-":
    gp = B_MINUS
elif letter == "C+":
    gp = C_PLUS
elif letter == "C":
    gp = C
elif letter == "C-":
    gp = C_MINUS
elif letter == "D+":
    gp = D_PLUS
elif letter == "D":
    gp = D
elif letter == "F":
    gp = F
else:
    gp = INVALID

# Выводим результат
if gp == INVALID:
    print("Введена некорректная оценка.")
else:
    print("Буквенная оценка", letter, "соответствует", gp, "баллам.")
```

Упражнение 54. Оценка работы

```
##
# Определение оценки работы сотрудников при помощи рейтингов от пользователя
#
RAISE_FACTOR = 2400.00
UNACCEPTABLE = 0
ACCEPTABLE = 0.4
MERITORIOUS = 0.6

# Запрашиваем у пользователя рейтинг
rating = float(input("Введите рейтинг: "))

# Классифицируем сотрудников
if rating == UNACCEPTABLE:
    performance = "низкий"
elif rating == ACCEPTABLE:
    performance = "удовлетворительный"
elif rating >= MERITORIOUS:
    performance = "высокий"
else:
    performance = ""
```

```
#Выводим результат
if performance == "":
    print("Введен ошибочный рейтинг.")
else:
    print("Основываясь на введенном рейтинге, ваш уровень: %s." % \
        performance)
print("Прибавка к зарплате составит: $%.2f." % \
    (rating * RAISE_FACTOR))
```

Необходимость заключать выражение `rating * RAISE_FACTOR` в последней строке кода в скобки объясняется тем, что операторы `%` и `*` имеют равный приоритет. Добавление скобок позволило сообщить Python, что сначала нужно выполнить математическую операцию, а затем – операцию форматирования.

Упражнение 58. Високосный год?

```
##
# Определяем, високосный заданный год или нет
#
# Запрашиваем у пользователя год
year = int(input("Введите год: "))

# Определяем, високосный или нет
if year % 400 == 0:
    isLeapYear = True
elif year % 100 == 0:
    isLeapYear = False
elif year % 4 == 0:
    isLeapYear = True
else:
    isLeapYear = False

# Отображаем результат
if isLeapYear:
    print(year, " - високосный год.")
else:
    print(year, " - невисокосный год.")
```

Упражнение 61. Действительный номерной знак машины?

```
## Определяем формат номерного знака. Всего допустимых формата два:
# 1) 3 буквы и 3 цифры
# 2) 4 цифры 3 буквы
# Запрашиваем номер у пользователя
plate = input("Введите номерной знак машины: ")
```

```
# Проверяем номерной знак. Необходимо проверить все 6 знаков для номера старого образца
# и 7 знаков – для нового
if len(plate) == 6 and \
  plate[0] >= "A" and plate[0] <= "Z" and \
  plate[1] >= "A" and plate[1] <= "Z" and \
  plate[2] >= "A" and plate[2] <= "Z" and \
  plate[3] >= "0" and plate[3] <= "9" and \
  plate[4] >= "0" and plate[4] <= "9" and \
  plate[5] >= "0" and plate[5] <= "9":
    print("Это номерной знак старого образца.")
elif len(plate) == 7 and \
  plate[0] >= "0" and plate[0] <= "9" and \
  plate[1] >= "0" and plate[1] <= "9" and \
  plate[2] >= "0" and plate[2] <= "9" and \
  plate[3] >= "0" and plate[3] <= "9" and \
  plate[4] >= "A" and plate[4] <= "Z" and \
  plate[5] >= "A" and plate[5] <= "Z" and \
  plate[6] >= "A" and plate[6] <= "Z":
    print("Это номерной знак нового образца.")
else:
    print("Неверный номерной знак.")
```

Упражнение 62. Играем в рулетку

```
##
# Определяем выпавший номер на рулетке и выигрыш
#
from random import randrange

# Симулируем запуск рулетки, используя число 37 для представления номера 00
value = randrange(0, 38)
if value == 37:
    print("Выпавший номер: 00...")
else:
    print("Выпавший номер: %d..." % value)

# Отображаем выигрыш для одного числа
if value == 37:
    print("Выигравшая ставка: 00")
else:
    print("Pay", value)

# Отображаем выигрыш по цветам
# В первой строке проверяем число на вхождение в ряд 1, 3, 5, 7 и 9
# Во второй строке проверяем число на вхождение в ряд 12, 14, 16 и 18
# В третьей строке проверяем число на вхождение в ряд 19, 21, 23, 25 и 27
# В четвертой строке проверяем число на вхождение в ряд 30, 32, 34 и 36
if value % 2 == 1 and value >= 1 and value <= 9 or \
  value % 2 == 0 and value >= 12 and value <= 18 or \
```

```
  value % 2 == 1 and value >= 19 and value <= 27 or \
  value % 2 == 0 and value >= 30 and value <= 36:
    print("Выигравшая ставка: красное")
elif value == 0 or value == 37:
    pass
else:
    print("Выигравшая ставка: черное")

# Отображаем выигрыш по чет/нечет
if value >= 1 and value <= 36:
    if value % 2 == 1:
        print("Выигравшая ставка: нечетное")
    else:
        print("Выигравшая ставка: четное")

# Отображаем выигрыш по низ/верх
if value >= 1 and value <= 18:
    print("Выигравшая ставка: от 1 до 18")
elif value >= 19 and value <= 36:
    print("Выигравшая ставка: от 19 до 36")
```

Тело блоков `if`, `elif` или `else` должно содержать по крайней мере одно выражение. В языке Python есть ключевое слово `pass`, которое можно использовать, когда требуется выражение, но никаких операций выполнять не нужно.

Глава 11

Повторения

Упражнение 66. Никаких центов

```
##
# Вычислить полную сумму покупки. Сумму для уплаты наличными округлить до
# ближайших пяти центов, поскольку одноцентовые монеты выведены из обращения
#
PENNIES_PER_NICKEL = 5
NICKEL = 0.05
```

> Хотя очень маловероятно, что в будущем в пятицентовой монете может оказаться больше пяти центов, мы все равно используем в программе переменные на случай, если когда-то нужно будет адаптировать ее для использования с другими номиналами монет.

```
# Собираем общую сумму
total = 0.00

# Запрашиваем цену первого товара как строку
line = input("Введите цену товара (пустая строка для выхода): ")

# Продолжаем запрашивать цены товаров, пока строка не будет оставлена пустой
while line != "":
    # Добавляем цену в общей сумме (после перевода ее в число с плавающей запятой)
    total = total + float(line)
    # Запрашиваем цену следующего товара
    line = input("Введите цену товара (пустая строка для выхода): ")

# Показываем полную сумму к оплате
print("Полная сумма к оплате: %.02f" % total)

# Считаем, сколько центов осталось бы, если бы мы оплатили всю покупку 5-центовыми
# монетами
rounding_indicator = total * 100 % PENNIES_PER_NICKEL
if rounding_indicator < PENNIES_PER_NICKEL / 2:
```

```
    # Если оставшаяся сумма центов меньше 2,5, округляем значение путем вычитания
    # полученного количества центов из общей суммы
    cash_total = total - rounding_indicator / 100
else:
    # Иначе добавляем 5 центов и затем вычитаем нужное количество центов
    cash_total = total + NICKEL - rounding_indicator / 100

# Выводим итоговую сумму для оплаты
print("Сумма для оплаты наличными: %.02f" % cash_total)
```

Упражнение 67. Найти периметр многоугольника

```
##
# Рассчитаем периметр многоугольника, построенного на основании координат точек,
# введенных пользователем. Пустая строка завершает ввод данных
#
from math import sqrt

# Переменная для хранения периметра многоугольника
perimeter = 0

# Запрашиваем координаты первой точки
first_x = float(input("Введите первую координату X: "))
first_y = float(input("Введите первую координату Y: "))

# Инициализируем координаты предыдущей точки начальными значениями
prev_x = first_x
prev_y = first_y

# Запрашиваем остальные координаты
line = input("Введите следующую координату X (Enter для окончания ввода): ")
while line != "":
    # Преобразуем координату X в число и запрашиваем координату Y
    x = float(line)
    y = float(input("Введите следующую координату Y: "))

    # Рассчитываем расстояние до предыдущей точки и добавляем к периметру
    dist = sqrt((prev_x - x) ** 2 + (prev_y - y) ** 2)
    perimeter = perimeter + dist

    # Устанавливаем значения предыдущих координат
    # для следующей итерации
    prev_x = x
    prev_y = y

    # Запрашиваем следующую координату X
    line = input("Введите следующую координату X (Enter для окончания ввода): ")

# Рассчитываем расстояние от последней точки до первой и добавляем к периметру
dist = sqrt((first_x - x) ** 2 + (first_y - y) ** 2)
```

Расстояние между точками на плоскости можно рассчитать по теореме Пифагора.

```
perimeter = perimeter + dist

# Отображаем результат
print("Периметр многоугольника равен", perimeter)
```

Упражнение 69. Билеты в зоопарк

```
##
# Рассчитать стоимость посещения зоопарка для группы
#
# Константы для хранения цен на разные категории билетов
BABY_PRICE = 0.00
CHILD_PRICE = 14.00
ADULT_PRICE = 23.00
SENIOR_PRICE = 18.00

# Сохраним как константы возрастные ограничения
BABY_LIMIT = 2
CHILD_LIMIT = 12
ADULT_LIMIT = 64

# Переменная для хранения общей суммы посещения
total = 0

# Читаем ввод пользователя до пустой строки
line = input("Введите возраст посетителя (пустая строка для окончания ввода): ")
while line != "":
    age = int(line)

    # Добавляем цену билета к общей сумме
    if age <= BABY_LIMIT:
        total = total + BABY_PRICE
    elif age <= CHILD_LIMIT:
        total = total + CHILD_PRICE
    elif age <= ADULT_LIMIT:
        total = total + ADULT_PRICE
    else:
        total = total + SENIOR_PRICE

    # Считываем возраст следующего посетителя
    line = input("Введите возраст посетителя (пустая строка для окончания ввода): ")

# Отображаем сумму группового посещения с правильным форматированием
print("Сумма посещения зоопарка для этой группы составит $%.2f" % total)
```

С нынешними правилами первый блок `if-elif-else` в нашей программе можно и не писать. Но он пригодится, если власти отменят бесплатное посещение зоопарка для маленьких деток.

Упражнение 70. Биты четности

```
##
# Рассчитать значение бита четности для набора из 8 бит, введенного пользователем
```

```
#
# Запрашиваем первые 8 бит
line = input("Введите 8 бит информации: ")
# Продолжаем цикл, пока пользователь не введет пустую строку
while line != "":
    # Убеждаемся в правильности ввода пользователя
    if line.count("0") + line.count("1") != 8 or len(line) != 8:
        # Выводим сообщение об ошибке
        print("Это не 8 бит… Попробуйте еще.")
    else:
        # Считаем единички
        ones = line.count("1")
```

Метод count возвращает количество вхождений указанной подстроки в строке, к которой применен.

```
        # Отображаем значение бита четности
        if ones % 2 == 0:
            print("Бит четности должен иметь значение 0.")
        else:
            print("Бит четности должен иметь значение 1.")

    # Считываем следующий ввод пользователя
    line = input("Введите 8 бит информации: ")
```

Упражнение 73. Код Цезаря

```
##
# Реализовать код Цезаря, в котором используются символы, сдвинутые
# на определенное количество позиций.
# Отрицательные значения сдвига можно использовать для декодирования.
#
# Запрашиваем у пользователя сообщение и сдвиг
message = input("Введите сообщение: ")
shift = int(input("Введите сдвиг: "))

# Обрабатываем каждый символ для создания зашифрованного сообщения
new_message = ""
for ch in message:
    if ch >= "a" and ch <= "z":
        # Обрабатываем букву в нижнем регистре, определяя ее позицию
        # в алфавите (0-25), вычисляя новую позицию и добавляя букву в сообщение
        pos = ord(ch) - ord("a")
        pos = (pos + shift) % 26
        new_char = chr(pos + ord("a"))
        new_message = new_message + new_char
```

Функция *ord* преобразует символ в целочисленную позицию в таблице ASCII. Функция *chr* возвращает символ в таблице ASCII по позиции, переданной в качестве аргумента.

```
    elif ch >= "A" and ch <= "Z":
        # Обрабатываем букву в верхнем регистре, определяя ее позицию
        # в алфавите (0-25), вычисляя новую позицию и добавляя букву в сообщение
        pos = ord(ch) - ord("A")
        pos = (pos + shift) % 26
        new_char = chr(pos + ord("A"))
        new_message = new_message + new_char
    else:
        # Если это не буква, просто сохраняем ее в сообщении
        new_message = new_message + ch
# Отображаем полученное сообщение
print("Новое сообщение", new_message)
```

Упражнение 75. Палиндром или нет?

```
##
# Определить, является ли введенная пользователем строка палиндромом
#

# Запрашиваем строку у пользователя
line = input("Введите строку: ")

# Предполагаем, что это палиндром, пока не доказано обратное
is_palindrome = True

# Сравниваем символы, начиная с двух сторон. Продолжаем, пока не достигнем середины или
# не поймем, что это не палиндром
i = 0
while i < len(line) / 2 and is_palindrome:
    # Если символы не равны, сразу понимаем, что это не палиндром
    if line[i] != line[len(line) - i - 1]:
        is_palindrome = False

    # Переходим к следующему символу
    i = i + 1

# Вывод результата
if is_palindrome:
    print(line, " - это палиндром")
else:
    print(line, " - это не палиндром")
```

Упражнение 77. Таблица умножения

```
##
# Вывести таблицу умножения от 1 до 10
#
MIN = 1
MAX = 10

# Строка заголовков
print("    ", end = "")
for i in range(MIN, MAX + 1):
    print("%4d" % i, end = "")
print()

# Выводим таблицу
for i in range(MIN, MAX + 1):
    print("%4d" % i, end = "")
    for j in range(MIN, MAX + 1):
        print("%4d" % (i * j), end = "")
    print()
```

Указание в качестве последнего аргумента функции print выражения end = “” позволяет избежать принудительного перевода строки.

Упражнение 79. Наибольший общий делитель

```
##
# Рассчитаем наибольший общий делитель для двух целых чисел с использованием цикла
#
# Запрашиваем у пользователя два целых числа
n = int(input("Введите первое целое число: "))
m = int(input("Введите второе целое число: "))

# Присваиваем d наименьшее из n и m
d = min(n, m)

# В цикле находим наибольший общий делитель для двух чисел
while n % d != 0 or m % d != 0:
    d = d - 1

# Выводим результат
print("Наибольший общий делитель для", n, "и", m, "равен", d)
```

Упражнение 82. Десятичное число в двоичное

```
##
# Перевести десятичное число в двоичное
#
NEW_BASE = 2

# Запрашиваем число для перевода у пользователя
num = int(input("Введите неотрицательное целое число: "))
```

```
# Будем сохранять результат в переменной result
result = ""
q = num
```

Алгоритм, предложенный здесь, выражен при помощи цикла *repeat-until* (повторяй-пока), но за неимением в Python такого типа циклов нам пришлось адаптировать алгоритм для работы с циклом `while`. Как пример, этого можно достигнуть путем дублирования тела цикла и размещения его непосредственно перед циклом.

```
# Пишем копию тела цикла вне самого цикла
r = q % NEW_BASE
result = str(r) + result
q = q // NEW_BASE

# Выполняем цикл, пока q не станет равен нулю
while q > 0:
    r = q % NEW_BASE
    result = str(r) + result
    q = q // NEW_BASE

# Отображаем результат
print(num, "в десятичной системе равно", result, "в двоичной.")
```

Упражнение 83. Максимальное число в последовательности

```
##
# Находим максимумы в случайном ряду из 100 целых чисел
# и считаем количество обновлений максимального значения
#
from random import randrange

NUM_ITEMS = 100

# Генерируем и выводим первое число
mx_value = randrange(1, NUM_ITEMS + 1)
print(mx_value)

# В этой переменной будем накапливать количество обновлений максимума
num_updates = 0

# Проходим по числам
for i in range(1, NUM_ITEMS):
    # Генерируем новое случайное число
    current = randrange(1, NUM_ITEMS + 1)

    # Если оно превышает текущий максимум...
```

```
    if current > mx_value:
        # Обновляем максимум и увеличиваем счетчик на единицу
        mx_value = current
        num_updates = num_updates + 1
        # Отображаем значение с пометкой
        print(current, "<== Обновление")
    else:
        # Отображаем значение
        print(current)

# Отображаем результаты
print("Максимальное значение в ряду:", mx_value)
print("Количество смен максимального значения:", num_updates)
```

Глава 12

Функции

Упражнение 88. Медиана трех значений

```
##
# Рассчитываем и выводим на экран медиану трех чисел, введенных пользователем
# В этой программе реализованы две техники вычисления медианы для демонстрации
# разных подходов к решению одной и той же задачи
#

## Рассчитываем медиану трех чисел при помощи блока if
# @param a – первое значение
# @param b – второе значение
# @param c – третье значение
# @return медиана чисел a, b и c
#
def median(a, b, c):
    if a < b and b < c or a > b and b > c:
        return b
    if b < a and a < c or b > a and a > c:
        return a
    if c < a and b < c or c > a and b > c:
        return c

## Рассчитываем медиану трех чисел при помощи функций min и max и капельки арифметики
# @param a – первое значение
# @param b – второе значение
# @param c – третье значение
# @return медиана чисел a, b и c
#
def alternateMedian(a, b, c):
    return a + b + c - min(a, b, c) - max(a, b, c)

# Выводим медиану чисел, введенных пользователем
def main():
    x = float(input("Введите первое число: "))
    y = float(input("Введите второе число: "))
```

Каждая функция, которую вы пишете, должна начинаться с комментария. Строки, начинающиеся с конструкции @param, используются для описания параметров. Строка, начинающаяся с @return, описывает возвращаемое значение.

Медиана трех чисел равна их сумме за вычетом минимального и максимального значений.

```
    z = float(input("Введите третье число: "))
    print("Медиана равна:", median(x, y, z))
    print("С помощью альтернативного метода:", \
        alternateMedian(x, y, z))

# Вызываем основную функцию
main()
```

Упражнение 90. Двенадцать дней Рождества

```
##
# Отображаем полный текст песни The Twelve Days of Christmas.
#
from int_ordinal import intToOrdinal
```

> Функция `intToOrdinal`, написанная вами для упражнения 89, импортируется здесь, чтобы не нужно было ее дублировать.

```
## Отображаем один куплет песни The Twelve Days of Christmas
# @param n - куплет для отображения
# @return (None)
def displayVerse(n):
    print("On the", intToOrdinal(n), "day of Christmas")
    print("my true love sent to me:")

    if n >= 12:
        print("Twelve drummers drumming,")
    if n >= 11:
        print("Eleven pipers piping,")
    if n >= 10:
        print("Ten lords a-leaping,")
    if n >= 9:
        print("Nine ladies dancing,")
    if n >= 8:
        print("Eight maids a-milking,")
    if n >= 7:
        print("Seven swans a-swimming,")
    if n >= 6:
        print("Six geese a-laying,")
    if n >= 5:
        print("Five golden rings,")
    if n >= 4:
        print("Four calling birds,")
    if n >= 3:
        print("Three French hens,")
    if n >= 2:
```

```
        print("Two turtle doves,")
    if n == 1:
        print("A", end=" ")
    else:
        print("And a", end=" ")
    print("partridge in a pear tree.")
    print()

# Отображаем все 12 куплетов песни
def main():
    for verse in range(1, 13):
        displayVerse(verse)

# Вызываем основную функцию
main()
```

Упражнение 93. Центрируем строку

```
##
# Центрируем строку в рамках заданного окна
#
WIDTH = 80

## Создаем новую строку, которая будет центрирована в окне заданной ширины
# @param s – строка для центрирования
# @param width – ширина окна, в котором будет центрирована строка
# @return копия строки s с ведущими пробелами для центрирования
def center(s, width):
# Если строка слишком длинная, возвращаем оригинал
    if width < len(s):
        return s

    # Вычисляем количество пробелов, необходимое для центрирования строки
    spaces = (width – len(s)) // 2
    result = " " * spaces + s
```

Оператор // используется, чтобы округлить результат деления до целого числа. Оператор / не может быть использован, поскольку возвращает число с плавающей точкой, а символ пробела можно поставить только целое количество раз.

```
    return result

# Демонстрируем центрирование строки
def main():
    print(center("Известная история", WIDTH))
    print(center("от:", WIDTH))
```

```
    print(center("Кого-то известного", WIDTH))
    print()
    print("Жили-были...")

# Call the main function
main()
```

Упражнение 95. Озаглавим буквы

```
##
# Заглавные буквы в строке
#

## Озаглавливаем нужные буквы в строке
# @param s - исходная строка для обработки
# @return новая строка
def capitalize(s):
    # Создаем копию исходной строки, в которой будем собирать итоговую строку
    result = s

    # Делаем заглавной первый непробельный символ в строке
    pos = 0
    while pos < len(s) and result[pos] == ' ':
        pos = pos + 1

    if pos < len(s):
        # Заменяем символ на заглавный, не затрагивая остальные символы в строке
        result = result[0 : pos] + result[pos].upper() + \
            result[pos + 1 : len(result)]
```

Использование двоеточия внутри квадратных скобок позволяет обратиться к подстроке, которая начинается с индекса, соответствующего числу до двоеточия, и заканчивается индексом, равным числу справа от двоеточия, не включая его.

```
    # Делаем заглавной первую букву, которая следует за точкой и
    # восклицательным или вопросительным знаком
    pos = 0
    while pos < len(s):
        if result[pos] == "." or result[pos] == "!" or \
            result[pos] == "?":
            # Переходим за знак ''.'', ''!'' or ''?''
            pos = pos + 1

            # Пропускаем пробелы
            while pos < len(s) and result[pos] == " ":
                pos = pos + 1
```

```
            # Если не достигли конца строки, меняем текущий символ на заглавную букву
            if pos < len(s):
                result = result[0 : pos] + \
                    result[pos].upper() + \
                    result[pos + 1 : len(result)]

        # Идем к следующему символу
        pos = pos + 1

    # Делаем заглавными буквы i, когда им предшествует пробел, а следом идет пробел,
    # точка, восклицательный или вопросительный знак либо апостроф
    pos = 1
    while pos < len(s) - 1:
        if result[pos - 1] == " " and result[pos] == "i" and \
            (result[pos + 1] == " " or result[pos + 1] == "." or \
                result[pos + 1] == "!" or result[pos + 1] == "?" or \
                result[pos + 1] == "'"):
            # Заменяем i на I, не затрагивая другие символы
            result = result[0 : pos] + "I" + \
                result[pos + 1 : len(result)]
        pos = pos + 1

    return result

# Демонстрируем работу функции
def main():
    s = input("Введите текст: ")
    capitalized = capitalize(s)
    print("Новая версия:", capitalized)

# Вызываем основную функцию
main()
```

Упражнение 96. Является ли строка целым числом?

```
##
# Определяем, представляет ли строка, введенная пользователем, целое число
#

## Посмотрим, включает ли строка целочисленное значение
# @param s - строка для проверки
# @return True, если это целое число. Иначе False.
#
def isInteger(s):
    # Удаляем пробелы в начале и конце строки
    s = s.strip()

    # Определяем, являются ли оставшиеся символы целыми числами
    if (s[0] == "+" or s[0] == "-") and s[1:].isdigit():
```

```
        return True
    if s.isdigit():
        return True
    return False
```

Метод isdigit возвращает True, если строка состоит как минимум из одного символа и все символы в ней являются цифрами.

```
# Демонстрируем работу функции isInteger
def main():
    s = input("Введите строку: ")
    if isInteger(s):
        print("Строка является целым числом.")
    else:
        print("Строка не является целым числом.")

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

Переменной __name__ автоматически присваивается значение при запуске программы на языке Python. При этом если файл запущен напрямую из Python, значение этой переменной будет равняться строке "__main__", а если импортирован в другую программу – имени модуля.

Упражнение 98. Простое число?

```
## Определяем, является ли число простым
# @param n - целое число для проверки
# @return True, если число простое, иначе False
def isPrime(n):
    if n <= 1:
        return False

    # Проверяем все числа от двух до n, не включая его, на деление n на них без остатка
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Если n % i == 0, значит, n без остатка делится на i, а следовательно, оно не простое.

```
# Определяем, является ли простым введенное пользователем число
def main():
    value = int(input("Введите целое число: "))
    if isPrime(value):
        print(value, "- простое число.")
    else:
        print(value, "не является простым числом.")

# Вызываем основную функцию, только если файл не импортирован
```

```
if __name__ == "__main__":
    main()
```

Упражнение 100. Случайный пароль

```
##
# Генерируем и отображаем случайный пароль, содержащий от 7 до 10 символов
#
from random import randint

SHORTEST = 7
LONGEST = 10
MIN_ASCII = 33
MAX_ASCII = 126

## Генерируем случайный пароль
# @return строка, содержащая случайный пароль
def randomPassword():
    # Выбираем случайную длину пароля
    randomLength = randint(SHORTEST, LONGEST)
    # Генерируем соответствующее количество случайных символов, добавляя их
к результату
    result = ""
    for i in range(randomLength):
        randomChar = chr(randint(MIN_ASCII, MAX_ASCII))
        result = result + randomChar
```

Функция chr принимает код ASCII в качестве единственного параметра и возвращает символ, соответствующий этому коду.

```
    # Возвращаем случайный пароль
    return result

# Генерируем и отображаем случайный пароль
def main():
    print("Ваш случайный пароль:", randomPassword())

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

Упражнение 102. Проверка пароля на надежность

```
##
# Проверка пароля на надежность
#
```

```
## Проверяем, является ли пароль надежным. Надежным будем считать пароль, в котором
# как минимум 8 символов, есть большие и маленькие буквы, а также цифры
# @param password - пароль для проверки
# @return True, если пароль надежен, иначе False
def checkPassword(password):
    has_upper = False
    has_lower = False
    has_num = False

    # Проверяем каждый символ в пароле
    for ch in password:
        if ch >= "A" and ch <= "Z":
            has_upper = True
        elif ch >= "a" and ch <= "z":
            has_lower = True
        elif ch >= "0" and ch <= "9":
            has_num = True

    # Если пароль отвечает всем четырем требованиям
    if len(password) >= 8 and has_upper and has_lower and has_num:
        return True

    # Если как минимум одно требование не соблюдено
    return False

# Демонстрируем работу функции
def main():
    p=input("Введите пароль: ")
    if checkPassword(p):
        print("Это надежный пароль.")
    else:
        print("Это ненадежный пароль.")

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

Упражнение 105. Произвольные системы счисления

```
##
# Переводим значение из одной системы счисления в другую. Диапазон систем - от 2 до 16
#
from hex_digit import *
```

Модуль `hex_digit` содержит функции `hex2int` и `int2hex`, которые мы написали, решая упражнение 104. Инструкция `import *` позволит загрузить все функции из модуля.

```
## Переводим число из десятичной системы в произвольную
# @param num - число в десятичной системе для преобразования
# @param new_base - основание для выходного результата
# @return строка в новой системе счисления
def dec2n(num, new_base):
    # Формируем представление числа в новой системе счисления, сохраняя в result
    result = ""
    q = num

    # Первый запуск тела будущего цикла
    r = q % new_base
    result = int2hex(r) + result
    q=q // new_base

    # Продолжаем цикл, пока q не станет равен нулю
    while q > 0:
        r = q % new_base
        result = int2hex(r) + result
        q = q // new_base

    # Возвращаем результат
    return result

## Переводим число из произвольной системы
# в десятичную
# @param num - число в системе по основанию b,
# сохраненное в виде строки
# @param b - основание преобразуемого числа
# @return число в десятичной системе счисления
def n2dec(num, b):
    decimal = 0

    # Обрабатываем каждую цифру по основанию b
    for i in range(len(num)):
        decimal = decimal * b
        decimal = decimal + hex2int(num[i])

    # Возвращаем результат
    return decimal

# Преобразуем число между произвольными системами счисления
def main():
    # Запрашиваем у пользователя исходную систему счисления и число
    from_base = int(input("Исходная система счисления (2-16): "))
    if from_base < 2 or from_base > 16:
        print("Допустимый диапазон систем счисления: от 2 до 16.")
        print("Выходим...")
        quit()

    from_num = input("Введите число по этому основанию: ")
```

Значение по основанию b должно быть представлено в виде строки, поскольку оно может содержать буквы, если основание счисления превышает 10.

```
    # Преобразуем в десятичное число и отображаем результат
    dec = n2dec(from_num, from_base)
    print("Результат: %d по основанию 10." % dec)

    # Преобразуем в число с новым основанием и отображаем результат
    to_base = int(input("Введите требуемую систему счисления (2-16): "))
    if to_base < 2 or to_base > 16:
        print("Допустимый диапазон систем счисления: от 2 до 16.")
        print("Выходим...")
        quit()

    to_num = dec2n(dec, to_base)
    print("Результат: %s по основанию %d." % (to_num, to_base))

# Вызов основной функции
main()
```

Упражнение 107. Максимальное сокращение дробей

```
##
# Максимальное сокращение дробей
#

## Вычислить наибольший общий делитель для двух целых чисел
# @param n - первое число (должно быть ненулевым)
# @param m - второе число (должно быть ненулевым)
# @return наибольший общий делитель двух целых чисел
def gcd(n, m):
    # Инициализируем d как меньшее значение из n и m
    d = min(n, m)
    # Используем цикл while для поиска наибольшего общего делителя n и m
    while n % d != 0 or m % d != 0:
        d = d - 1
    return d
```

Для достижения цели функция gcd использует цикл. Также существует простой и элегантный способ определения наибольшего общего делителя двух чисел при помощи рекурсии. Эта концепция будет описана в упражнении 174.

```
## Сокращаем дробь до предела
# @param num - числитель дроби
# @param den - знаменатель дроби (должен быть ненулевым)
# @return числитель и знаменатель сокращенной дроби
def reduce(num, den):
    # Если числитель равен нулю, сокращенная дробь будет равна 0/1
    if num == 0:
```

```
        return (0, 1)

    # Находим наибольший общий делитель числителя и знаменателя
    g = gcd(num, den)

    # Делим числитель и знаменатель на НОД и возвращаем результат
    return (num // g, den // g)
```

В функции reduce мы использовали оператор //, чтобы вернуть целочисленные значения числителя и знаменателя.

```
# Запрашиваем дробь у пользователя и отображаем ее максимально сокращенный вариант
def main():
    # Запрашиваем числитель и знаменатель у пользователя
    num = int(input("Введите числитель дроби: "))
    den = int(input("Введите знаменатель дроби: "))

    # Вычисляем сокращенную дробь
    (n, d) = reduce(num, den)

    # Выводим результат
    print("Дробь %d/%d может быть сокращена до %d/%d." % (num, den, n, d))

# Вызов основной функции
main()
```

Упражнение 108. Переводим меры

```
##
# Преобразуем меры объема ингредиентов в рецептах с целью их более лаконичного
# выражения
# Например, 59 чайных ложек можно сократить до 1 стакана, 3 столовых ложек
# и 2 чайных ложек.
#
TSP_PER_TBSP = 3
TSP_PER_CUP = 48

##
# Преобразуем меры объема ингредиентов в рецептах с целью их более лаконичного
# выражения
# @param num - количество единиц объема для преобразования
# @param unit - единица измерения (''cup'', ''tablespoon'' или ''teaspoon'')
# @return строка, представляющая меры в сокращенной форме
def reduceMeasure(num, unit):
    # Приводим единицу измерения к нижнему регистру
    unit = unit.lower()
```

Единицы измерения приводятся к нижнему регистру путем вызова метода lower и сохранения результата в ту же переменную. Это позволит пользователю вводить меру в любом регистре.

```
# Вычислим объем в чайных ложках
if unit == "teaspoon" or unit == "teaspoons":
    teaspoons = num
elif unit == "tablespoon" or unit == "tablespoons":
    teaspoons = num * TSP_PER_TBSP
elif unit == "cup" or unit == "cups":
    teaspoons = num * TSP_PER_CUP

# Преобразуем объем в чайных ложках в другие единицы измерения
cups = teaspoons // TSP_PER_CUP
teaspoons = teaspoons - cups * TSP_PER_CUP
tablespoons = teaspoons // TSP_PER_TBSP
teaspoons = teaspoons - tablespoons * TSP_PER_TBSP

# Создаем строковую переменную для хранения результата
result = ""

# Добавляем количество стаканов к результату (если надо)
if cups > 0:
    result = result + str(cups) + " cup"
    # Множественное число
    if cups > 1:
        result = result + "s"

# Добавляем количество столовых ложек к результату (если надо)
if tablespoons > 0:
    # Добавляем запятую, если нужно
    if result != "":
        result = result + ", "

    result = result + str(tablespoons) + " tablespoon"
    # Множественное число
    if tablespoons > 1:
        result = result + "s"

# Добавляем количество чайных ложек к результату (если надо)
if teaspoons > 0:
    # Добавляем запятую, если нужно
    if result != "":
        result = result + ", "
        result = result + str(teaspoons) + " teaspoon"
        # Множественное число
        if teaspoons > 1:
```

```
                result = result + "s"

    # Обрабатываем ноль
    if result == "":
        result = "0 teaspoons"

    return result
```

В эту программу мы включили сразу несколько вызовов функции для охвата большого числа разнообразных мер.

```
# Демонстрируем работу функции reduceMeasure путем нескольких обращений
def main():
    print("59 teaspoons is %s." % reduceMeasure(59, "teaspoons"))
    print("59 tablespoons is %s." % \
    reduceMeasure(59, "tablespoons"))
    print("1 teaspoon is %s." % reduceMeasure(1, "teaspoon"))
    print("1 tablespoon is %s." % reduceMeasure(1, "tablespoon"))
    print("1 cup is %s." % reduceMeasure(1, "cup"))
    print("4 cups is %s." % reduceMeasure(4, "cups"))
    print("3 teaspoons is %s." % reduceMeasure(3, "teaspoons"))
    print("6 teaspoons is %s." % reduceMeasure(6, "teaspoons"))
    print("95 teaspoons is %s." % reduceMeasure(95, "teaspoons"))
    print("96 teaspoons is %s." % reduceMeasure(96, "teaspoons"))
    print("97 teaspoons is %s." % reduceMeasure(97, "teaspoons"))
    print("98 teaspoons is %s." % reduceMeasure(98, "teaspoons"))
    print("99 teaspoons is %s." % reduceMeasure(99, "teaspoons"))

# Вызов основной функции
main()
```

Упражнение 109. Магические даты

```
##
# Определяем все магические даты в XX веке
#
from days_in_month import daysInMonth

## Определяем, является ли дата магической
# @param day - день в дате
# @param month - месяц в дате
# @param year - год в дате
# @return True, если дата является магической, иначе False
def isMagicDate(day, month, year):
    if day * month == year % 100:
        return True
```

Выражение `year % 100` позволяет перейти к представлению года из двух цифр.

```
    return False

# Находим и отображаем все магические даты XX века
def main():
    for year in range(1900, 2000):
        for month in range(1, 13):
            for day in range(1, daysInMonth(month, year) + 1):
                if isMagicDate(day, month, year):
                    print("%02d/%02d/%04d - магическая дата." % (day, month, year))

# Вызов основной функции
main()
```

Глава 13

Списки

Упражнение 110. Порядок сортировки

```
##
# Выводим числа, введенные пользователем, в порядке возрастания
#
# Начинаем с пустого списка
data = []

# Считываем значения и добавляем их в список, пока пользователь не введет ноль
num = int(input("Введите целое число (0 для окончания ввода): "))
while num != 0:
    data.append(num)
    num = int(input("Введите целое число (0 для окончания ввода): "))

# Сортируем значения
data.sort()
```

Вызов метода `sort` применительно к списку переставляет значения в нем, располагая их в нужном нам порядке. В данном случае этот способ подходит, поскольку нам нет необходимости сохранять копию исходного списка. Для создания копии исходного списка с отсортированными элементами можно воспользоваться функцией `sorted`. Вызов этой функции не оказывает влияния на порядок следования элементов в исходном списке, а значит, ее стоит использовать, когда в дальнейшем вам пригодятся обе версии списка: исходный и отсортированный в нужном вам порядке.

```
# Выводим числа в порядке возрастания
print("Введенные числа в порядке возрастания:")
for num in data:
    print(num)
```

Упражнение 112. Удаляем выбросы

```
##
# Удаляем выбросы из набора данных
#
```

```
## Удаляем выбросы из списка значений
# @param data – список значений для обработки
# @param num_outliers – количество наименьших и наибольших элементов для удаления
# @return копия исходного списка с отсортированными значениями и
# удаленными наименьшими и наибольшими элементами
def removeOutliers(data, num_outliers):
    # Создаем копию списка с отсортированными значениями
    retval = sorted(data)

    # Удаляем num_outliers наибольших значений
    for i in range(num_outliers):
        retval.pop()

    # Удаляем num_outliers наименьших значений
    for i in range(num_outliers):
        retval.pop(0)

    # Возвращаем результат
    return retval

# Запрашиваем данные у пользователя и удаляем по два наибольших и наименьших значения
def main():
    # Запрашиваем данные у пользователя, пока он не оставит ввод пустым
    values = []
    s=input("Введите значение (Enter для окончания ввода): ")
    while s != "":
        num = float(s)
        values.append(num)
        s = input("Введите значение (Enter для окончания ввода): ")

    # Отображаем результат или соответствующее сообщение об ошибке
    if len(values) < 4:
        print("Вы ввели недостаточное количество чисел.")
    else:
        print("Список с удаленными выбросами: ", \
                removeOutliers(values, 2))
        print("Исходный список: ", values)

# Вызов основной функции
main()
```

Наибольшие и наименьшие значения в списке можно было удалить и в одном цикле. В данном случае используется два цикла, чтобы решение было более понятным.

Упражнение 113. Избавляемся от дубликатов

```
##
# Считываем ряд слов, введенных пользователем, и выводим их без дубликатов
# в том же порядке, в котором они были введены
#

# Запрашиваем слова у пользователя и сохраняем их в список
words = []
```

```
word = input("Введите слово (Enter для окончания ввода): ")
while word != "":
    # Добавляем слово в список, только если
    # оно уже не присутствует в нем
    if word not in words:
        words.append(word)

    # Запрашиваем следующее слово у пользователя
    word = input("Введите слово (Enter для окончания ввода): ")

# Отображаем уникальные слова
for word in words:
    print(word)
```

Выражения `word not in words` и `not (word in words)` эквивалентны.

Упражнение 114. Отрицательные, положительные и нули

```
##
# Запрашиваем коллекцию целых чисел у пользователя. Отображаем сначала отрицательные,
# затем нули и после этого положительные
#

# Создаем три списка для хранения отрицательных, нулевых и положительных значений
negatives = []
zeros = []
positives = []
```

В данном решении используется отдельный список для хранения введенных пользователем нулей. Но в этом нет особой необходимости, поскольку нули все одинаковые. Достаточно было бы хранить количество введенных пользователем нулей и при выводе отображать их столько, сколько надо.

```
# Запрашиваем числа у пользователя, помещая их в соответствующие списки
line = input("Введите целое число (Enter для окончания ввода): ")
while line != "":
    num = int(line)
    if num < 0:
        negatives.append(num)
    elif num > 0:
        positives.append(num)
    else:
        zeros.append(num)

    # Запрашиваем следующее число у пользователя
    line = input("Введите целое число (Enter для окончания ввода): ")

# Выводим сначала отрицательные числа, затем нули и после этого положительные
print("Введенные числа: ")
for n in negatives:
```

```
    print(n)
for n in zeros:
    print(n)
for n in positives:
    print(n)
```

Упражнение 116. Совершенные числа

```
##
# Целое число n называется совершенным, если сумма всех его собственных делителей
# равна самому числу n
# Покажем все совершенные числа от 1 до LIMIT.
#
from proper_divisors import properDivisors

LIMIT = 10000

## Определяем, является ли число совершенным
# @param n - число, которое необходимо проверить на совершенство
# @return True, если число совершенно, иначе False
def isPerfect(n):
    # Получим список собственных
    # делителей числа
    divisors = properDivisors(n)

    # Рассчитываем их сумму
    total = 0
    for d in divisors:
        total = total + d

    # Определяем, является ли число совершенным, и возвращаем результат
    if total == n:
        return True
    return False

# Отображаем все совершенные числа от 1 до LIMIT.
def main():
    print("Совершенные числа от 1 до", LIMIT, ":")
    for i in range(1, LIMIT + 1):
        if isPerfect(i):
            print(" ", i)

#Call the main function
main()
```

Сумма элементов списка также может быть вычислена при помощи функции `sum` языка Python. Это позволит избежать написания цикла и сократить операцию до одной строки.

Упражнение 120. Форматирование списка

```
##
# Отображаем перечень введенных слов через запятую и с союзом "и" между
# последними двумя словами
#
```

```
## Форматируем список с запятыми и союзом "и"
#@param items - список элементов для форматирования
#@return строка с установленными правилами форматирования
def formatList(items):
    # Отдельно рассматриваем пустой список и список из одного элемента
    if len(items) == 0:
        return "<пусто>"
    if len(items) == 1:
        return str(items[0])

    # Идем по всем элементам списка, за исключением двух последних
    result = ""
    for i in range(0, len(items) - 2):
        result = result + str(items[i]) + ", "
```

Каждый введенный элемент мы явным образом преобразуем в строку путем вызова функции `str` перед выполнением форматирования. Это позволит функции `formatList` корректно обрабатывать не только строковые элементы, но и числовые.

```
    # Добавляем к строке два последних элемента, разделенных союзом "и"
    result = result + str(items[len(items) - 2]) + " и "
    result = result + str(items[len(items) - 1])
    # Возвращаем результат
    return result

# Запрашиваем у пользователя слова и форматируем их
def main():
    # Запрашиваем у пользователя слова, пока не будет пропущена строка ввода
    items = []
    line = input("Введите слово (Enter для окончания ввода): ")

    while line != "":
        items.append(line)
        line = input("Введите слово (Enter для окончания ввода): ")

    # Форматируем и отображаем результат
    print("Введенные элементы: %s." % formatList(items))

# Вызов основной функции
main()
```

Упражнение 121. Случайные лотерейные номера

```
##
# Собираем уникальные случайные номера для лотерейного билета
#
from random import randrange
```

```
MIN_NUM = 1
MAX_NUM = 49
NUM_NUMS = 6
```

Использование констант поможет быстро адаптировать программу для любой лотереи.

```
# Используем список для хранения номеров лотерейного билета
ticket_nums = []

# Генерируем NUM_NUMS случайных, но уникальных значений
for i in range(NUM_NUMS):
    # Генерируем номер, которого еще нет в списке
    rand = randrange(MIN_NUM, MAX_NUM + 1)
    while rand in ticket_nums:
        rand = randrange(MIN_NUM, MAX_NUM + 1)

    # Добавляем номер к билету
    ticket_nums.append(rand)

# Сортируем номера по возрастанию и отображаем их
ticket_nums.sort()
print("Номера вашего билета: ", end="")
for n in ticket_nums:
    print(n, end=" ")
print()
```

Упражнение 125. Тасуем колоду карт

```
##
# Создаем колоду карт и перетасовываем ее
#
from random import randrange

## Генерируем стандартную колоду карт с четырьмя мастями и 13 номиналами в каждой
# @return список карт с каждой картой, представленной двумя символами
def createDeck():
    # Создаем список для хранения карт
    cards = []

    # Проходим по всем мастям и номиналам
    for suit in ["s", "h", "d", "c"]:
        for value in ["2", "3", "4", "5", "6", "7", "8", "9", \
                      "T", "J", "Q", "K", "A"]:

            # Генерируем карту и добавляем ее в колоду
            cards.append(value + suit)

    # Возвращаем целую колоду
    return cards

## Тасуем колоду, переданную в функцию в качестве параметра
# @param cards - список карт для тасования
```

```
# @return (None)
def shuffle(cards):
    # Проходим по картам
    for i in range(0, len(cards)):
        # Выбираем случайный индекс между текущим индексом и концом списка
        other_pos = randrange(i, len(cards))

        # Меняем местами текущую карту со случайно выбранной
        temp = cards[i]
        cards[i] = cards[other_pos]
        cards[other_pos] = temp

# Отображаем колоду до и после тасования
def main():
    cards = createDeck()
    print("Исходная колода карт: ")
    print(cards)
    print()

    shuffle(cards)
    print("Перетасованная колода карт: ")
    print(cards)

# Вызываем основную функцию, только если программа не была импортирована как модуль
if __name__ == "__main__":
    main()
```

Упражнение 128. Подсчитать элементы в списке

```
##
# Подсчитываем количество элементов в списке, больших или равных
# заданному минимуму и меньших заданного максимума
#

## Определяем, сколько элементов в списке больше или равны
# заданному минимуму и меньше заданного максимума
# @param data - список значений для обработки
# @param mn - минимальная граница
# @param mx - максимальная граница
# @return количество элементов e, отвечающее условию mn <= e < mx
def countRange(data, mn, mx):
    # Подсчитываем количество элементов в списке из указанного диапазона
    count = 0
    for e in data:
        # Проверяем каждый элемент
        if mn <= e and e < mx:
            count = count + 1

    # Возвращаем результат
    return count
```

```
# Демонстрируем работу функции countRange
def main():
    data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    # Случай, когда несколько элементов входят в диапазон
    print("Подсчитываем количество элементов в списке [1..10] между 5 и 7...")
    print("Результат: %d Ожидание: 2" % countRange(data, 5, 7))

    # Случай, когда все элементы входят в диапазон
    print("Подсчитываем количество элементов в списке [1..10] между -5 и 77...")
    print("Результат: %d Ожидание: 10" % countRange(data, -5, 77))

    # Случай, когда ни один из элементов не входит в диапазон
    print("Подсчитываем количество элементов в списке [1..10] между 12 и 17...")
    print("Результат: %d Ожидание: 0" % countRange(data, 12, 17))

    # Случай, когда список пуст
    print("Подсчитываем количество элементов в списке [] между 0 и 100...")
    print("Результат: %d Ожидание: 0" % countRange([], 0, 100))

    # Случай, когда в списке есть дубликаты
    data = [1, 2, 3, 4, 1, 2, 3, 4]
    print("Подсчитываем количество элементов в списке", data, "между 2 и 4...")
    print("Результат: %d Ожидание: 4" % countRange(data, 2, 4))

# Вызов основной программы
main()
```

Упражнение 129. Разбиение строки на лексемы

```
##
# Разбиение строки, содержащей математическое выражение, на лексемы
#

## Преобразуем математическое выражение в список лексем
# @param s - строка для разбора
# @return список лексем строки s или пустой список, если возникла ошибка
def tokenList(s) :
    # Удаляем все пробелы из строки s
    s = s.replace(" ", "")

    # Проходим по символам в строке, определяя лексемы и добавляя их к списку
    tokens = []
    i = 0
    while i < len(s):
        # Обрабатываем односимвольные лексемы: *, /, ^, ( и )
        if s[i] == "*" or s[i] == "/" or s[i] == "^" or \
                s[i] == "(" or s[i] == ")" or s[i] == "+" or s[i] == "-":
            tokens.append(s[i])
            i = i + 1
```

```
        # Обрабатываем числа без лидирующих + или -
        elif s[i] >= "0" and s[i] <= "9":
            num = ""
            # Добавляем символы в лексему, пока они представляют собой цифры
            while i < len(s) and s[i] >= "0" and s[i] <= "9":
                num = num + s[i]
                i = i + 1
            tokens.append(num)

        # Наличие других символов означает недействительность выражения.
        # Возвращаем пустой список для сигнализации возникновения ошибки
        else:
            return []

    return tokens

# Запрашиваем выражение у пользователя, разбиваем на лексемы и отображаем результат
def main():
    exp = input("Введите математическое выражение: ")
    tokens = tokenList(exp)
    print("Лексемы:", tokens)

# Вызываем основную функцию, только если программа не была импортирована как модуль
if __name__ == "__main__":
    main()
```

Упражнение 130. Унарные и бинарные операторы

```
##
# Устанавливаем разницу между унарными и бинарными операторами + и -
#
from token_list import tokenList

## Определяем появление унарных операторов + и - в списке лексем и
# меняем их на u+ и u- соответственно
# @param tokens - список лексем, который может содержать унарные операторы + и -
# @return список лексем с заменой унарных операторов + и - на u+ и u-
def identifyUnary(tokens):
    retval = []

    # Обрабатываем каждую лексему в списке
    for i in range(len(tokens)):
        # Если первая лексема - это + или -, то это унарный оператор
        if i == 0 and (tokens[i] == "+" or tokens[i] == "-"):
            retval.append("u" + tokens[i])
        # Если это лексема + или -, а предыдущая лексема являлась
        # оператором или открывающей скобкой, то это унарный оператор
        elif i>0and (tokens[i] == "+" or tokens[i] == "-") and \
                (tokens[i-1] == "+" or tokens[i-1] == "-" or
                tokens[i-1] == "*" or tokens[i-1] == "/" or
```

```
                tokens[i-1] == "("):
            retval.append("u" + tokens[i])
        # Любая другая лексема свидетельствует о том, что это бинарный оператор,
        # так что он добавляется к списку без изменений
        else:
            retval.append(tokens[i])

    # Возвращаем новый список лексем с измененными унарными операторами
    return retval

# Демонстрируем выполнение пометки унарных операторов
def main():
    # Запрашиваем выражение у пользователя, разбиваем на лексемы и отображаем результат
    exp = input("Введите математическое выражение: ")
    tokens = tokenList(exp)
    print("Лексемы:", tokens)
    # Идентифицируем список унарных операторов
    marked = identifyUnary(tokens)
    print("С помеченными унарными операторами: ", marked)

# Вызываем основную функцию, только если программа не была импортирована как модуль
if __name__ == "__main__":
    main()
```

Упражнение 134. Все подсписки заданного списка

```
##
# Находим все подсписки в заданном списке
#

## Создаем список из всех подсписков заданного списка
# @param data – список, в котором выполняется поиск подсписков
# @return список из всех подсписков исходного списка
def allSublists(data):
    # Начинаем с добавления пустого списка
    sublists = [[]]

    # Генерируем подсписки длиной от 1 до len(data)
    for length in range(1, len(data) + 1):
        # Генерируем подсписки начиная с каждого индекса
        for i in range(0, len(data) – length + 1):
            # Добавляем найденный подсписок к общему списку
            sublists.append(data[i : i + length])

    # Возвращаем результат
    return sublists

# Демонстрируем работу функции allSublists
def main():
    print("Подсписки []: ")
```

Список с пустым списком внутри обозначается как [[]].

```
    print(allSublists([]))

    print("Подсписки [1]: ")
    print(allSublists([1]))

    print("Подсписки [1, 2]: ")
    print(allSublists([1, 2]))

    print("Подсписки [1, 2, 3]: ")
    print(allSublists([1, 2, 3]))

    print("Подсписки [1, 2, 3, 4]: ")
    print(allSublists([1, 2, 3, 4]))

# Вызов основной программы
main()
```

Упражнение 135. Решето Эратосфена

```
##
# Определяем все простые числа от 2 до значения, введенного пользователем,
# при помощи алгоритма "Решето Эратосфена"
#
# Запрашиваем у пользователя конечное значение
limit = int(input("Вывести простые числа вплоть до какого значения? "))

# Создаем список для чисел от 0 до limit
nums = []
for i in range(0, limit + 1):
    nums.append(i)

# "Вычеркиваем" единицу, заменяя ее на ноль
nums[1] = 0

# Вычеркиваем числа, кратные всем найденным простым числам
p = 2
while p < limit:
    # Вычеркиваем все числа, кратные p, но не его само
    for i in range(p * 2, limit + 1, p):
        nums[i] = 0

    # Находим следующее "невычеркнутое" число
    p = p + 1
    while p < limit and nums[p] == 0:
        p = p + 1

# Отображаем результат
print("Простые числа вплоть до", limit, ":")
for i in nums:
    if nums[i] != 0:
        print(i)
```

Глава 14

Словари

Упражнение 136. Поиск по значению

```
##
# Проводим поиск всех ключей в словаре по заданному значению
#

## Поиск в словаре по значению
# @param data - словарь для поиска
# @param value - искомое значение
# @return список (возможно, пустой) ключей, соответствующих искомому значению
def reverseLookup(data, value):
    # Создаем список ключей для заданного значения
    keys = []

    # Проверяем каждый ключ и добавляем в список,
    # если он соответствует искомому значению
    for key in data:
        if data[key] == value:
            keys.append(key)

    # Возвращаем список ключей
    return keys

# Демонстрируем работу reverseLookup
def main():
    # Словарь соответствий французских слов английским
    frEn = {"le" : "the", "la" : "the", "livre" : "book", \
            "pomme" : "apple"}

    # Показываем работу функции reverseLookup для трех случаев:
    # для множества ключей, одного ключа и отсутствия ключей
    print("Перевод английского слова 'the' на французский: ", \
            reverseLookup(frEn, "the"))
    print("Ожидаемый результат: ['le', 'la']")
    print()
    print("Перевод английского слова 'apple' на французский: ", \
```

Каждый ключ в словаре должен быть уникальным. При этом значения могут повторяться. Таким образом, поиск в словаре по значению может привести к результату, содержащему от нуля до множества ключей.

```
            reverseLookup(frEn, "apple"))
    print("Ожидаемый результат: ['pomme']")
    print()
    print("Перевод английского слова 'asdf' на французский: ", \
            reverseLookup(frEn, "asdf"))
    print("Ожидаемый результат: []")

# Вызываем основную функцию, только если файл не был импортирован в качестве модуля
if __name__ == "__main__":
    main()
```

Упражнение 137. Две игральные кости

```
##
# Симуляция многократного выбрасывания двух игральных костей и сравнение
# полученных результатов с ожидаемыми
#
from random import randrange

NUM_RUNS = 1000
D_MAX = 6

## Симуляция выбрасывания двух шестигранных игральных костей
# @return общее количество очков, выпавших на двух костях
def twoDice():
    # Две кости
    d1 = randrange(1, D_MAX + 1)
    d2 = randrange(1, D_MAX + 1)

    # Возвращаем сумму
    return d1 + d2

# Симулируем многократное выбрасывание костей и отображаем результат
def main():
    # Составим словарь для ожидаемых значений
    expected = {2: 1/36, 3: 2/36, 4: 3/36, 5: 4/36, 6: 5/36, \
            7: 6/36, 8: 5/36, 9: 4/36, 10: 3/36, \
            11: 2/36, 12: 1/36}

    # Составим словарь для хранения результатов выбрасывания костей
    counts = {2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, \
            8: 0, 9: 0, 10: 0, 11: 0, 12: 0}
```

Словари изначально инициализированы ключами от 2 до 12. В словаре `expected` значения заполнены в соответствии с теорией вероятностей, тогда как в словаре `counts` изначально стоят нули, которые будут меняться в процессе симуляции выбрасывания костей.

```
    # Проведение NUM_RUNS симуляций и подсчет результатов
    for i in range(NUM_RUNS):
        t = twoDice()
        counts[t] = counts[t] + 1

    # Сравниваем ожидаемые результаты с реальными
    print("Всего  Реальный  Ожидаемый")
    print("        процент    процент")
    for i in sorted(counts.keys()):
        print("%5d %9.2f %10.2f" % \
            (i, counts[i] / NUM_RUNS * 100, expected[i] * 100))

# Вызов основной функции
main()
```

Упражнение 142. Уникальные символы

```
##
# Считаем уникальные символы в строке при помощи словаря
#

# Запрашиваем строку у пользователя
s = input("Введите строку: ")

# Добавляем каждый символ в словарь со значением True. После окончания процедуры
# количество ключей в словаре и будет отражать число уникальных символов
characters = {}
for ch in s:
    characters[ch] = True
```

Каждому ключу в словаре должно соответствовать значение. Но в нашем примере эти значения никогда не будут использованы, так что мы решили применить `True`. Вместо него мы могли использовать любое другое значение.

```
# Отображаем результат
print("Строка содержит", len(characters), \
        "уникальных символов.")
```

Функция `len` возвращает количество ключей в словаре.

Упражнение 143. Анаграммы

```
##
# Определяем, являются ли два введенных пользователем слова анаграммами
#
```

```
## Рассчитываем распределение частоты появления символов в строке
# @param s – строка для обработки
# @return словарь с количеством символов в строке
def characterCounts(s):
    # Создаем пустой словарь
    counts = {}

    # Обновляем словарь для каждого символа в строке
    for ch in s:
        if ch in counts:
            counts[ch] = counts[ch] + 1
        else:
            counts[ch] = 1

    # Возвращаем результат
    return counts

# Определяем, являются ли строки, введенные пользователем, анаграммами
def main():
    # Запрашиваем строки у пользователя
    s1 = input("Введите первую строку: ")
    s2 = input("Введите вторую строку: ")

    # Вызываем функцию для двух введенных строк
    counts1 = characterCounts(s1)
    counts2 = characterCounts(s2)

    # Выводим результат
    if counts1 == counts2:
        print("Введенные строки являются анаграммами.")
    else:
        print("Введенные строки не являются анаграммами.")
```

Два словаря считаются равными, если содержат одинаковое количество ключей и те же самые ассоциированные с ними значения.

```
# Вызов основной функции
main()
```

Упражнение 145. Эрудит

```
##
# Используем словарь для подсчета количества очков за собранное слово в Эрудите
#

# Создаем словарь с соответствием букв и очков за них
points = {"A": 1, "B": 3, "C": 3, "D": 2, "E": 1, "F": 4, \
```

```
            "G": 2, "H": 4, "I": 1, "J": 2, "K": 5, "L": 1, \
            "M": 3, "N": 1, "O": 1, "P": 3, "Q": 10, "R": 1, \
            "S": 1, "T": 1, "U": 1, "V": 4, "W": 4, "X": 8,
            "Y": 4, "Z": 10}

# Запрашиваем у пользователя слово
word = input("Введите слово: ")

# Считаем количество очков
uppercase = word.upper()
score = 0
for ch in uppercase:
    score = score + points[ch]

# Выводим результат
print(word, "оценивается в", score, "очков.")
```

Введенное слово мы переводим в верхний регистр, чтобы не ограничивать пользователя в выборе регистра при вводе слова. Этого результата можно было добиться и путем добавления строчных букв к словарю.

Упражнение 146. Карточка лото

```
##
# Создадим и отобразим случайную карточку лото
#
from random import randrange

NUMS_PER_LETTER = 15

## Создание случайной карточки для игры в лото
# @return словарь с ключами, представляющими буквы B, I, N, G и O,
# и списком номеров под каждой буквой
def createCard():
    card = {}

    # Диапазон целых чисел для букв
    lower = 1
    upper = 1 + NUMS_PER_LETTER

    # Для каждой из пяти букв
    for letter in ["B", "I", "N", "G", "O"]:
        # Создаем пустой список для буквы
        card[letter] = []

        # Генерируем случайные номера, пока не наберется пять уникальных
        while len(card[letter]) != 5:
            next_num = randrange(lower, upper)
            # Убеждаемся, что не храним дубликаты номеров
            if next_num not in card[letter]:
                card[letter].append(next_num)

        # Обновляем диапазон номеров для следующей буквы
        lower = lower + NUMS_PER_LETTER
```

```
        upper = upper + NUMS_PER_LETTER

    # Возвращаем сгенерированную карточку
    return card

## Выводим отформатированную карточку лото
# @param card – карточка лото для отображения
# @return (None)
def displayCard(card):
    # Заголовки
    print("B  I  N  G  O")

    # Отображаем номера
    for i in range(5):
        for letter in ["B", "I", "N", "G", "O"]:
            print("%2d " % card[letter][i], end="")
        print()

# Создаем случайную карточку для игры в лото и отображаем ее
def main():
    card = createCard()
    displayCard(card)

# Вызываем основную функцию, только если файл не импортирован
if __name__ == "__main__":
    main()
```

Глава 15

Файлы и исключения

Упражнение 149. Отображаем заголовок файла

```
##
# Показываем первые 10 строк файла, имя которого передано в качестве аргумента
командной строки
#
import sys

NUM_LINES = 10

# Проверяем, что программе был передан только один аргумент командной строки
if len(sys.argv) != 2:
    print("Передайте имя файла в качестве аргумента командной строки.")
    quit()

try:
    # Открываем файл на чтение
    inf = open(sys.argv[1], "r")

    # Читаем первую строку из файла
    line = inf.readline()

    # Продолжаем цикл, пока не прочитаем 10 строк или не дойдем до конца файла
    count = 0
    while count < NUM_LINES and line != "":
        # Удаляем символ конца строки и увеличиваем счетчик
        line = line.rstrip()
        count = count + 1

        # Отображаем строку
        print(line)

        # Читаем следующую строку из файла
        line = inf.readline()

        # Закрываем файл
        inf.close()
```

По вызову функции `quit` программа мгновенно завершается.

```
except IOError:
    # Отображаем ошибку, если с чтением из файла возникли проблемы
    print("Ошибка при доступе к файлу.")
```

Упражнение 150. Отображаем конец файла

```
##
# Показываем первые 10 строк файла, имя которого передано в качестве аргумента
командной строки
#
import sys

NUM_LINES = 10

# Проверяем, что программе был передан только один аргумент командной строки
if len(sys.argv) != 2:
    print("Передайте имя файла в качестве аргумента командной строки.")
    quit()

try:
    # Открываем файл на чтение
    inf = open(sys.argv[1], "r")

    # Читаем файл, сохраняя NUM_LINES последних строк
    lines = []
    for line in inf:
        # Добавляем последнюю прочитанную строку к концу списка
        lines.append(line)
        # Если у нас накопилось больше NUM_LINES строк, удаляем самую старую
        if len(lines) > NUM_LINES:
            lines.pop(0)

    # Закрываем файл
    inf.close()

except:
    print("Ошибка при доступе к файлу.")
    quit()

# Отображаем последние строки из файла
for line in lines:
    print(line, end="")
```

Упражнение 151. Сцепляем файлы

```
##
# Сцепляем два или более файлов и отображаем результат
#
import sys
```

```
# Убедимся, что хотя бы один параметр был передан в качестве аргумента командной строки
if len(sys.argv) == 1:
    print("Нужно передать программе хотя бы один аргумент.")
    quit()

# Обрабатываем все файлы, имена которых были переданы в качестве аргументов
for i in range(1, len(sys.argv)):
    fname = sys.argv[i]
    try:
        # Открываем текущий файл на чтение
        inf = open(fname, "r")

        # Отображаем файл
        for line in inf:
            print(line, end="")

        # Закрываем файл
        inf.close()

    except:
        # Отображаем предупреждение, но не завершаем выполнение программы
        print("Невозможно открыть/отобразить файл", fname)
```

Элемент с индексом 0 в списке sys.argv является ссылкой на исполняемый файл Python. Поэтому наш цикл for начинает анализировать параметры с индекса 1.

Упражнение 156. Сумма чисел

```
##
# Подсчитаем сумму переданных пользователем чисел, не считая нечислового ввода
#

# Запрашиваем у пользователя первое число
line = input("Введите число: ")
total = 0

# Продолжаем запрашивать числа, пока пользователь не оставит ввод пустым
while line != "":
    try:
        # Пытаемся конвертировать значение в число
        num = float(line)
        # Если все прошло успешно, прибавляем введенное число к общей сумме
        total = total + num
        print("Сумма сейчас составляет", total)
    except ValueError:
        # Отображаем предупреждение и переходим к следующему вводу
        print("Это было не число")

    # Запрашиваем следующее число
    line = input("Введите число: ")

# Отображаем сумму
print("Общая сумма:", total)
```

Упражнение 158. Удаляем комментарии

```
##
# Удаляем все комментарии из файла Python (за исключением ситуаций, когда
# символ комментария указан в середине строки)
#

# Запрашиваем у пользователя имя исходного файла и открываем его
try:
    in_name = input("Введите имя файла Python: ")
    inf = open(in_name, "r")

except:
    # Выводим сообщение и завершаем работу программы в случае возникновения ошибки
    print("При открытии файла возникла проблема.")
    print("Завершение работы программы...")
    quit()

# Запрашиваем у пользователя имя итогового файла и открываем его
try:
    out_name = input("Введите имя нового файла: ")
    outf = open(out_name, "w")

except:
    # Закрываем исходный файл, показываем сообщение об ошибке и завершаем программу
    inf.close()
    print("С создаваемым файлом возникла проблема.")
    print("Завершение работы программы...")
    quit()

try:
    # Читаем строки из исходного файла, избавляем их от комментариев и
    # сохраняем в новом файле
    for line in inf:
        # Находим позицию символа комментария (-1, если его нет)
        pos = line.find("#")
        # Если комментарий есть, создаем
        # строковый срез со всеми знаками до него
        # и сохраняем обратно в переменную line
        if pos > -1:
            line = line[0 : pos]
            line = line + "\n"

        # Записываем потенциально измененную
        # строку в новый файл
        outf.write(line)

    # Закрываем файлы
    inf.close()
    outf.close()
```

Позиция символа комментария сохраняется в переменной `pos`. Так что выражение `line[0 : pos]` указывает на все символы в строке до знака комментария, не включая его.

```
except:
    # Выводим сообщение об ошибке, если что-то пошло не так
    print("При обработке файла возникла проблема.")
    print("Завершаем программу...")
```

Упражнение 159. Случайный пароль из двух слов

```
##
# Генерируем пароль путем сцепления двух случайных слов. Длина пароля должна составлять
# от 8 до 10 символов, а каждое слово должно быть длиной минимум 3 символа
#
from random import randrange

WORD_FILE = "../Data/words.txt"
```

Пароль, который мы создаем, должен содержать от 8 до 10 символов. Поскольку минимальная длина слова составляет 3 символа, а в пароле должно быть два слова, получается, что в пароле не может присутствовать слово длиной более 7 символов.

```
# Читаем все слова из файла, оставляя только те из них, длина которых варьируется
# от 3 до 7 символов, и сохраняем их в список
words = []
inf = open(WORD_FILE, "r")
for line in inf:
    # Удаляем символ новой строки
    line = line.rstrip()

    # Оставляем слова длиной от 3 до 7 символов
    if len(line) >= 3 and len(line) <= 7:
        words.append(line)

# Закрываем файл
inf.close()

# Случайным образом выбираем первое слово для пароля
first = words[randrange(0, len(words))]
first = first.capitalize()

# Выбираем второе слово для пароля до тех пор, пока оно не будет подходить по размеру
password = first
while len(password) < 8 or len(password) > 10:
    second = words[randrange(0, len(words))]
    second = second.capitalize()
    password = first + second

# Отображаем случайный пароль
print("Случайный пароль:", password)
```

Упражнение 162. Книги без буквы E

```
##
# Выводим процент слов, использующих все буквы в алфавите
# Также найдем букву, реже остальных встречающуюся в словах
#
WORD_FILE = "../Data/words.txt"

# Создадим словарь со счетчиком слов, содержащих каждую букву.
# Для каждой буквы инициализируем счетчик нулем
counts = {}
for ch in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    counts[ch] = 0

# Открываем файл, обрабатываем каждое слово и обновляем словарь со счетчиками
num_words = 0
inf = open(WORD_FILE, "r")
for word in inf:
    # Переводим слово в верхний регистр и избавляемся от символа новой строки
    word = word.upper().rstrip()
    # Перед обновлением словаря нужно создать список уникальных букв в слове.
    # Иначе мы будем увеличивать счетчик для повторяющихся букв в слове.
    # Также будем игнорировать небуквенные символы
    unique = []
    for ch in word:
        if ch not in unique and ch >= "A" and ch <= "Z":
            unique.append(ch)

    # Увеличим счетчики для всех букв в списке уникальных символов
    for ch in unique:
        counts[ch] = counts[ch] + 1

    # Увеличиваем количество обработанных слов
    num_words = num_words + 1

# Закрываем файл
inf.close()

# Выводим результат для каждой буквы. Параллельно определяем, какая буква
# встречается в словах наиболее редко, чтобы вывести ее отдельно.
smallest_count = min(counts.values())
for ch in sorted(counts):
    if counts[ch] == smallest_count:
        smallest_letter = ch
    percentage = counts[ch] / num_words * 100
    print(ch, "встречается в %.2f процентах слов" % percentage)

# Отображаем самую редко используемую букву в словах
print()
print("Буква, от которой легче всего будет избавиться:", smallest_letter)
```

Упражнение 163. Популярные детские имена

```
##
# Отображаем мужские и женские имена, бывшие самыми популярными как минимум
# в одном году с 1900-го по 2012-й
#

FIRST_YEAR = 1900
LAST_YEAR = 2012

## Загружаем первую строку из файла, извлекаем имя и добавляем его к списку имен,
# если его там еще нет
# @param fname - имя файла, из которого будут считываться данные
# @param names - список, к которому будем добавлять данные (если они отсутствуют)
# @return (None)
def LoadAndAdd(fname, names):
    # Открываем файл, читаем первую строку и извлекаем имя
    inf = open(fname, "r")
    line = inf.readline()
    inf.close()
    parts = line.split()
    name = parts[0]

    # Добавляем имя в список, если оно там еще не присутствует
    if name not in names:
        names.append(name)

# Отображаем мужские и женские имена, бывшие самыми популярными как минимум
# в одном году с 1900-го по 2012-й
def main():
    # Создаем списки для хранения самых популярных имен
    girls = []
    boys = []

    # Обрабатываем все годы из диапазона, читая первые строки из файлов с мужскими
и женскими именами
    for year in range(FIRST_YEAR, LAST_YEAR + 1):
        girl_fname = "../Data/BabyNames/" + str(year) + \
                     "_GirlsNames.txt"
        boy_fname = "../Data/BabyNames/" + str(year) + \
                     "_BoysNames.txt"
```

В данном решении я предположил, что файл с именами лежит не в той же папке, где файл Python. Если ваши файлы находятся в той же директории, часть пути к файлу `../Data/BabyNames/` нужно пропустить.

```
        LoadAndAdd(girl_fname, girls)
```

```
        LoadAndAdd(boy_fname, boys)

    # Выводим списки
    print("Самые популярные имена девочек:")
    for name in girls:
        print(" ", name)
    print()

    print("Самые популярные имена мальчиков: ")
    for name in boys:
        print(" ", name)

# Вызов основной функции
main()
```

Упражнение 167. Проверяем правильность написания

```
##
# Находим и отображаем все слова в файле, написанные неправильно
#
from only_words import onlyTheWords
import sys

WORDS_FILE = "../Data/words.txt"

# Убедимся, что программе передано допустимое количество аргументов командной строки
if len(sys.argv) != 2:
    print("При вызове программы должен быть указан один аргумент.")
    print("Завершаем программу...")
    quit()

# Открываем файл. Выходим, если возникла ошибка
try:
    inf = open(sys.argv[1], "r")
except:
    print("Ошибка при открытии файла '%s' на чтение. Завершаем программу..." % \
            sys.argv[1])
    quit()

# Загружаем все слова в словарь. В значения ставим 0, но использовать его не будем
valid = {}
words_file = open(WORDS_FILE, "r")

for word in words_file:
    # Переводим слово в нижний регистр и удаляем символ новой строки
    word = word.lower().rstrip()

    # Добавляем слово в словарь
    valid[word] = 0

words_file.close()
```

В данном решении используются словари со словами в виде ключей и неиспользуемыми значениями. В целом эффективнее в этой ситуации будет использовать наборы (set), если вы знакомы с подобной структурой данных. Списки использовать не рекомендуется из-за их медлительности при поиске элементов.

```
# Читаем все строки из файла, добавляя слова с ошибками в соответствующий список
misspelled = []
for line in inf:
    # Избавляемся от знаков препинания при помощи функции из упражнения 117
    words = onlyTheWords(line)
    for word in words:
        # Если слово написано неправильно и отсутствует в списке, добавляем его туда
        if word.lower() not in valid and word not in misspelled:
            misspelled.append(word)

# Закрываем анализируемый файл
inf.close()

# Отображаем слова с ошибками или сообщение об их отсутствии
if len(misspelled) == 0:
    print("Все слова написаны правильно.")
else:
    print("Следующие слова написаны неправильно:")
    for word in misspelled:
        print(" ", word)
```

Упражнение 169. Редактирование текста в файле

```
##
# Редактируем файл, удаляя в нем служебные слова. Отредактированная версия
# записывается в новый файл
#
# Заметьте, что в этой программе не выполняются проверки на ошибки
# и она регистрозависимая
#

# Запрашиваем у пользователя имя файла для редактирования и открываем его
inf_name = input("Введите имя файла для редактирования: ")
inf = open(inf_name, "r")

# Запрашиваем у пользователя имя файла со служебными словами и открываем его
sen_name = input("Введите имя файла со служебными словами: ")
sen = open(sen_name, "r")

# Загружаем все служебные слова в список
words = []
line = sen.readline()
```

```
while line != "":
    line = line.rstrip()
    words.append(line)
    line = sen.readline()

# Закрываем файл со служебными словами
sen.close()
```

Файл со служебными словами может быть закрыт в середине программы, поскольку все слова из него уже считаны в список.

```
# Запрашиваем у пользователя имя нового файла и открываем его
outf_name = input("Введите имя нового файла: ")
outf = open(outf_name, "w")

# Считываем все строки из исходного файла. Заменяем все служебные слова на звездочки.
# Пишем строки в новый файл.
line = inf.readline()
while line != "":
    # Ищем и заменяем служебные слова. Количество звездочек соответствует
    # длине исходного слова
    for word in words:
        line = line.replace(word, "*" * len(word))

    # Пишем измененную строку в новый файл
    outf.write(line)

    # Читаем следующую строку из исходного файла
    line = inf.readline()

# Закрываем исходный и новый файлы
inf.close()
outf.close()
```

Упражнение 170. Пропущенные комментарии

```
##
# Находим и отображаем имена функций Python, которым не предшествует строка
с комментарием
#
from sys import argv

# Проверяем аргументы командной строки
if len(argv) == 1:
    print("По крайней мере одно имя файла должно присутствовать в качестве", \
            "аргумента командной строки.")
    print("Завершаем программу...")
```

```
    quit()

# Обрабатываем все файлы, имена которых переданы в виде аргументов командной строки
for fname in argv[1 : len(argv)]:
    # Попытка обработать файл
    try:
        inf = open(fname, "r")
        # Двигаясь по файлу, нам важно хранить копию предыдущей строки, чтобы
        # можно было проверить, начинается ли она с символа комментария.
        # Также нам необходимо считать строки в файле
        prev = " "
        lnum = 1
```

Переменная `prev` должна быть инициализирована строкой длиной как минимум в один символ. Иначе программа закроется аварийно, если в первой строке анализируемого файла будет начинаться определение функции.

```
        # Считываем все строки из файла
        for line in inf:
            # Если нашли функцию без комментария
            if line.startswith("def ") and prev[0] != "#":
                # Ищем первую открывающую скобку в строке, чтобы считать имя функции
                bracket_pos = line.index("(")
                name = line[4 : bracket_pos]

                # Отображаем информацию о недокументированной функции
                print("%s строка %d: %s" % (fname, lnum, name))
            # Сохраняем текущую строку и обновляем счетчик
            prev = line
            lnum = lnum + 1
        # Закрываем текущий файл
        inf.close()
    except:
        print("Возникла проблема с файлом '%s'." % fname)
        print("Идем к следующему файлу...")
```