**3 Лабораторная работа. Циклы**

Python умеет работать как минимум с двумя видами данных — числами и строками. Числа записываются последовательностью цифр, также перед числом может стоять знак минус, а строки записываются в одинарных кавычках. 2 и ‘2’ — это разные объекты, первый объект — число, а второй — строка. Операция + для целых чисел и для строк работает по-разному: для чисел это сложение, а для строк — конкатенация.

Кроме целых чисел есть и другой класс чисел: действительные (вещественные числа), представляемые в виде десятичных дробей. Они записываются с использованием десятичной точки, например, 2.0.

Определить тип объекта можно при помощи функции type:

**>>>** type(2)

<class 'int'>

**>>>** type('2')

<class 'str'>

**>>>** type(2.0)

<class 'float'>

Обратите внимание — type является функцией, аргументы функции указываются в скобках после ее имени.

### Операции с числами

Вот список основных операций для чисел:

* A+B — сумма;
* A-B — разность;
* A\*B — произведение;
* A/B — частное;
* A\*\*B — возведение в степень.

Полезно помнить, что квадратный корень из числа x — это x\*\*0.5, а корень степени n — это x\*\*(1/n).

Есть также унарный вариант операции -, то есть операция с одним аргументом. Она возвращает число, противоположное данному. Например: -A.

В выражении может встречаться много операций подряд. Как в этом случае определяется порядок действий? Например, чему будет равно 1+2\*3\*\*1+1? В данном случае ответ будет 8, так как сначала выполняется возведение в степень, затем — умножение, затем — сложение.

Более общие правила определения приоритетов операций такие:

1. Выполняются возведения в степень **справа налево**, то есть 3\*\*3\*\*3 это 3²⁷.
2. Выполняются унарные минусы (отрицания).
3. Выполняются умножения и деления слева направо. Операции умножения и деления имеют одинаковый приоритет.
4. Выполняются сложения и вычитания слева направо. Операции сложения и вычитания имеют одинаковый приоритет.

### Операции над строками

* A+B — конкатенация;
* A\*n — повторение n раз, значение n должно быть целого типа.

### Ветвление

Ветвление (или условная инструкция) в Python имеет следующий синтаксис:

if Условие:

Блок\_инструкций\_1

else:

Блок\_инструкций\_2

Блок\_инструкций\_1 будет выполнен, если Условие истинно. Если Условие ложно, будет выполнен Блок\_инструкций\_2.

В условной инструкции может отсутствовать слово else и последующий блок. Такая инструкция называется неполным ветвлением. Например, если дано число x и мы хотим заменить его на абсолютную величину x, то это можно сделать следующим образом:

if x < 0:

x = -x

print(x)

В этом примере переменной x будет присвоено значение -x, но только в том случае, когда x<0. А вот инструкция print(x) будет выполнена всегда, независимо от проверяемого условия.

Для выделения блока инструкций, относящихся к инструкции if или else в языке Python используются отступы. Все инструкции, которые относятся к одному блоку, должны иметь равную величину отступа, то есть одинаковое число пробелов в начале строки. Рекомендуется использовать отступ в 4 пробела.

### Вложенные условные инструкции

Внутри условных инструкций можно использовать любые инструкции языка Python, в том числе и условную инструкцию. Вложенное ветвление — после одной развилки в ходе исполнения программы появляется другая развилка. При этом вложенные блоки имеют больший размер отступа (например, 8 пробелов).

Примере программы, которая по данным ненулевым числам x и y определяет, в какой из четвертей координатной плоскости находится точка (x,y):

x = int(input())

y = int(input())

if x > 0:

if y > 0: *# x>0, y>0*

print("Первая четверть")

else: *# x>0, y<0*

print("Четвертая четверть")

else:

if y > 0: *# x<0, y>0*

print("Вторая четверть")

else: *# x<0, y<0*

print("Третья четверть")

В этом примере использованы  комментарии – текст, который интерпретатор игнорирует. Комментариями в Python является символ # и весь текст после этого символа до конца строки.

### Операторы сравнения

Как правило, в качестве проверяемого условия используется результат вычисления одного из следующих операторов сравнения:

| **Оператор** | **Значение** |
| --- | --- |
| < | Меньше — условие верно, если первый операнд меньше второго. |
| > | Больше — условие верно, если первый операнд больше второго. |
| <= | Меньше или равно — условие верно, если первый операнд меньше или равен второму. |
| >= | Больше или равно — условие верно, если первый операнд больше или равен второму. |
| == | Равенство. Условие верно, если два операнда равны. |

Например, условие (x \* x < 1000) означает «значение x \* x меньше 1000», а условие (2 \* x != y) означает «удвоенное значение переменной x не равно значению переменной y».

Операторы сравнения в можно объединять в цепочки, например, a == b == c или 1 <= x <= 10.

### Тип данных bool

Операторы сравнения возвращают значения специального логического типа bool. Значения логического типа могут принимать одно из двух значений: True (истина) или False (ложь). Если преобразовать логическое True к типу int, то получится 1, а преобразование False даст 0. При обратном преобразовании число 0 преобразуется в False, а любое ненулевое число в True. При преобразовании str в bool пустая строка преобразовывается в False, а любая непустая строка в True.

### Каскадные условные инструкции

Пример программы, определяющий четверть координатной плоскости, можно переписать используя “каскадную” последовательность операцией if... elif... else:

x = int(input())

y = int(input())

if x > 0 and y > 0:

print("Первая четверть")

elif x < 0 and y > 0:

print("Вторая четверть")

elif x < 0 and y < 0:

print("Третья четверть")

elif x > 0 and y < 0:

print("Четвертая четверть")

else:

print("Точка находится на осях или в центре координат.")

В такой конструкции условия if, ..., elif проверяются по очереди, выполняется блок, соответствующий первому из истинных условий. Если все проверяемые условия ложны, то выполняется блок else, если он присутствует. Обратите внимание, что таким образом мы чётче видим условия наступления случаев (нет "призраков"), а также отлавливаем ситуацию, когда точка не находится ни в одной из четвертей.

### Цикл while

Цикл while («пока») позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл while используется, когда невозможно определить точное значение количества проходов исполнения цикла.

Синтаксис цикла while в простейшем случае выглядит так:

while Условие:

Блок\_инструкций

При выполнении цикла while сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла while. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

Например, следующий фрагмент программы напечатает на экран всех целые числа, не превосходящие n:

a = 1

while a <= n:

print(a)

a += 1

Общая схема цикла while в данном случае для перебора всех подходящих значений такая:

a = начальное\_значение

**while** а\_является\_подходящим\_числом:

обработать\_a

перейти\_к\_следующему\_a

Выводем все степени двойки, не превосходящие числа n:

a = 1

while a <= n:

print(a)

a \*= 2

### Цикл for

Цикл for может быть использован как более краткая альтернатива циклу while.

Для последовательного перебора целых чисел из диапазона [0; n) можно использовать цикл for:

for i in range(10):

print(i)

Этот код по выполняемым действиям полностью соответствуют циклу while:

i = 0

while i < 10:

print(i)

i += 1

Можно задавать начальные и конечные значения для переменной цикла, а также шаг:

for i in range(20, 10, -2):

print(i)

Аналогичный цикл while

i = 20

while i > 10:

print(i)

i -= 2

## Черепаха

Стандартная библиотека Python содержит модуль turtle, предназначенный для обучения программированию. Этот модуль содержит **набор** функций, позволяющих управлять черепахой. Черепаха умеет выполнять небольшой набор команд, а именно:

| **Команда** | **Значение** |
| --- | --- |
| forward(X) | Пройти вперёд X пикселей |
| backward(X) | Пройти назад X пикселей |
| left(X) | Повернуться налево на X градусов |
| right(X) | Повернуться направо на X градусов |
| penup() | Не оставлять след при движении |
| pendown() | Оставлять след при движении |
| shape(X) | Изменить значок черепахи (“arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”) |
| stamp() | Нарисовать копию черепахи в текущем месте |
| color() | Установить цвет |
| begin\_fill() | Необходимо вызвать перед рисованием фигуры, которую надо закрасить |
| end\_fill() | Вызвать после окончания рисования фигуры |
| width() | Установить толщину линии |
| goto(x, y) | Переместить черепашку в точку (x, y) |

Например, следующая программа рисует букву S:

import turtle

turtle.shape('turtle')

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.left(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

turtle.right(90)

turtle.forward(50)

**Задание на 3 лабораторную работу:**

1. Напишите конструкцию, с помощью которой на экране будет распечатана оценка, полученная студентом за тест в зависимости от полученных правильных ответов, при этом

* более 90% правильных ответов студент получает «5»,
* за 70-89% правильных ответов студент получает «4»,
* за 60-70% правильных ответов студент получает «3»,
* менее 60% правильных ответов студент получает неудовлетворительно.

Количество правильных ответов и общее количество заданий в тесте вводятся с клавиатуры на усмотрение преподавателя.

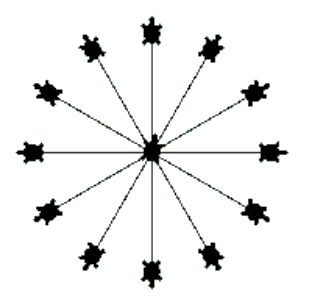
1. Напишите конструкцию, которая принимает на вход количество заказанных бутылей воды в компании в течение месяца и рассчитывает общую стоимость, которую нужно оплатить в конце месяца. При этом стоимость зависит от заказанного объема:

|  |  |
| --- | --- |
| **Количество заказанных бутылей в течение месяца** | **Стоимость одной бутыли** |
| менее 10 | 303 |
| 10 – 20 | скидка 7% |
| более 20 | скидка 12% |

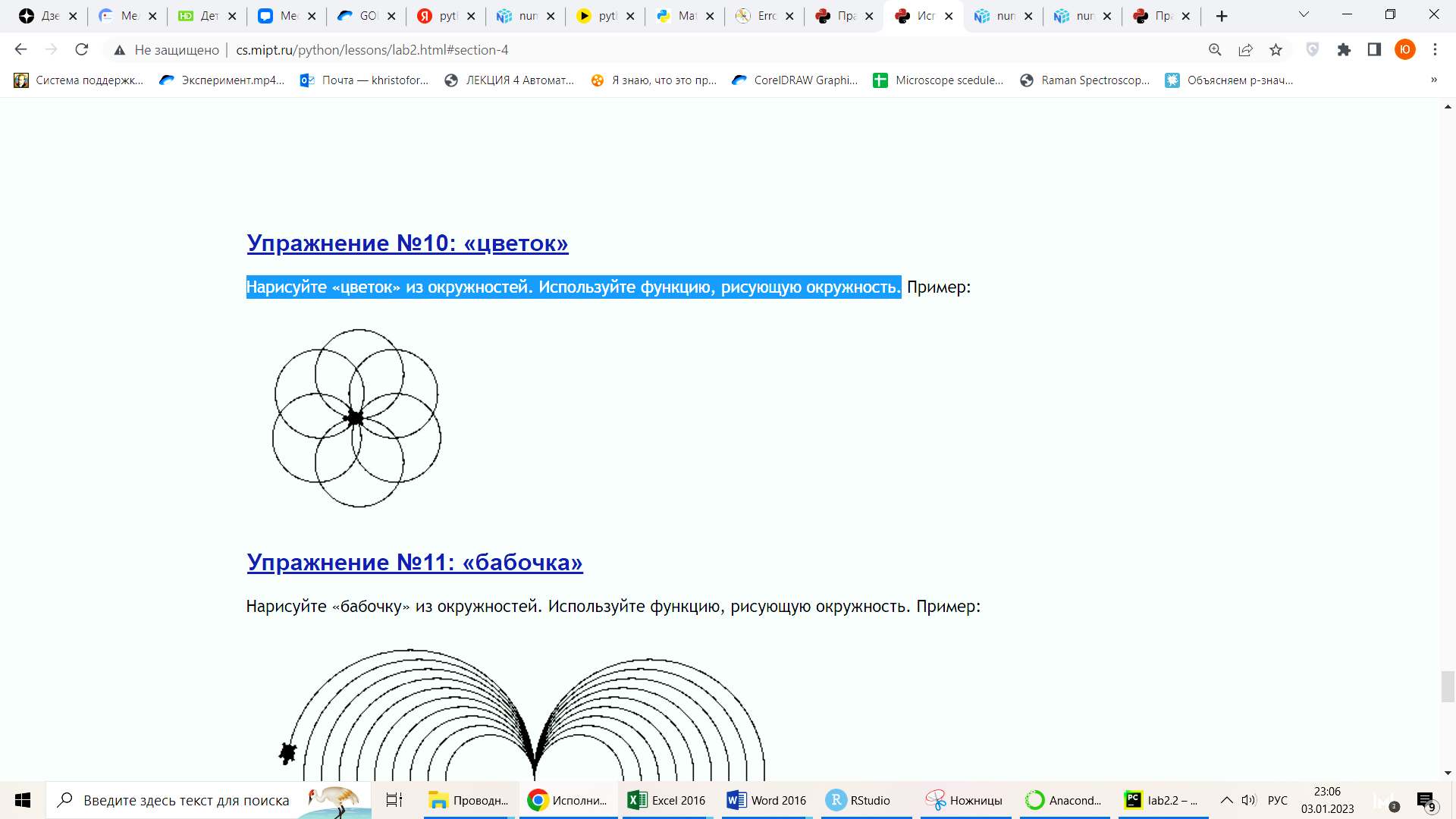
Ответ округлите до десяток рублей

**Используя циклы и модуль turtle выполните следующие задания:**

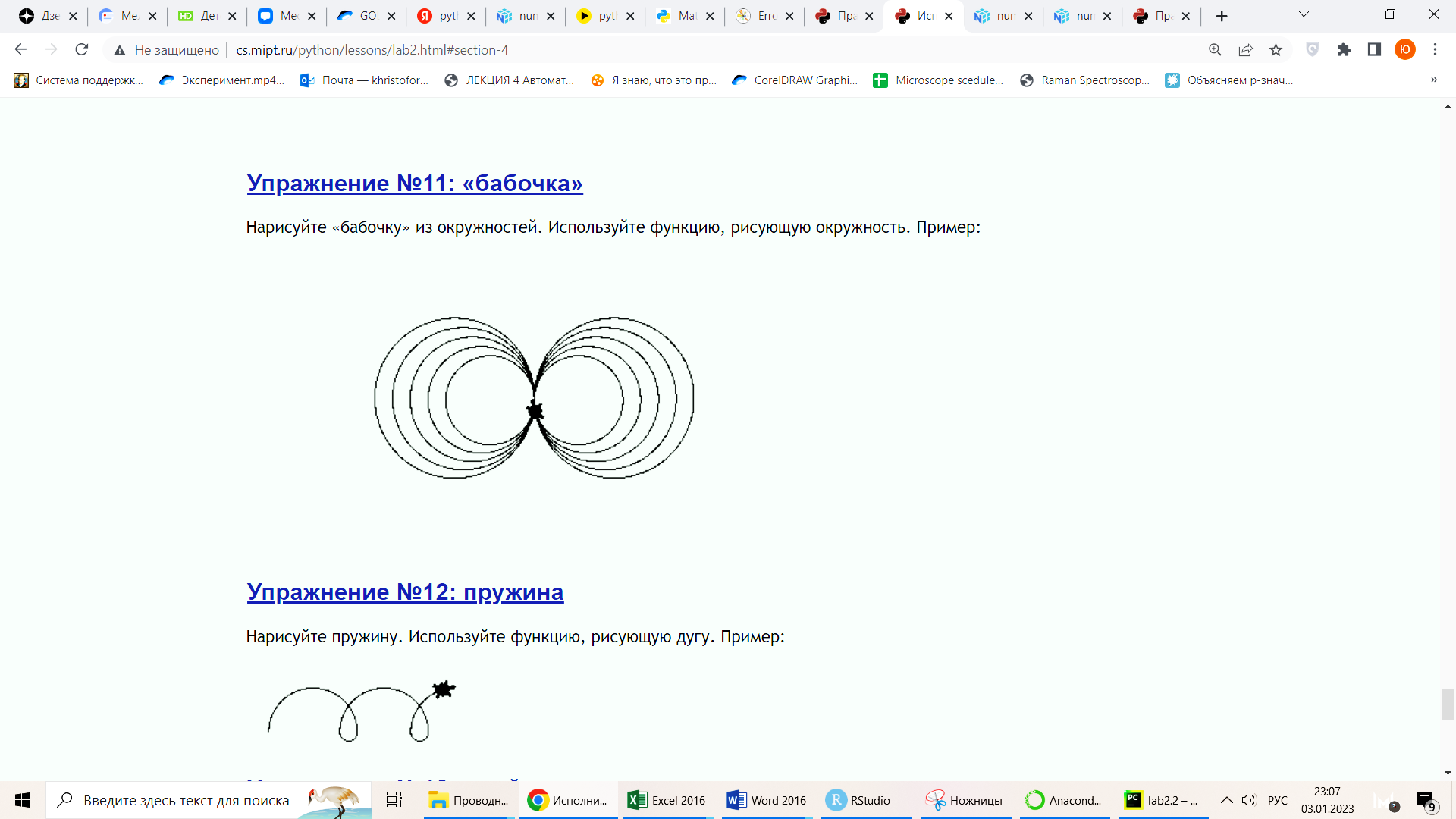
1. Нарисуйте квадрат с помощью черепашки.
2. Нарисуйте окружность с помощью черепашки. Можно учитывать правило, что правильный многоугольник с большим количеством сторон и углов, будет являться окружностью.
3. Нарисуйте 10 вложенных квадратов с помощью черепашки.
4. Нарисуйте паука с n лапами. Например, n = 12.



1. Нарисуйте квадратную спираль с помощью черепашки.
2. Нарисуйте цветок из окружностей с помощью черепашки. Пример:



1. Нарисуйте бабочку из окружностей с помощью черепашки. Пример:



1. Нарисуйте две звезды: одну с 5 вершинами, другую — с 11.