МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СЕВЕРО-КАВКАЗСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ

Е. Х. Бежанова
А. С.-А. Джатдоев

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

Лабораторных практикум для обучающихся на 1 курса направление подготовки 01.03.02 Прикладная математика и информатика очной формы обучения

г. Черкесск 2024 г

УДК 004.42
ББК 32.973
Б 38

Рассмотрено на заседании кафедры «Математика»
Протокол № 1 от 31 08. 2023 г.
Рекомендовано к изданию редакционно-издательским советом СКГА
Протокол №26 от 29.09.2023 г.

Рецензенты:
Темирова Л.Г. – к. ф.-м. н., доцент
Шапошникова О.И.– к. ф.-м. н., доцент

Б38 **Бежанова, Е.Х.** Процедурное программирование: лабораторный практикум для обучающихся 1 курса направления подготовки 01.03.02 Прикладная математика информатика / Е.Х. Бежанова, А.С.-А. Джатдоев.– Черкесск: СКГА, 2024. – 32с.

Данный лабораторный практикум предназначен для обучающихся направление подготовки 01.03.02 Прикладная математика, и информатика содержит краткий теоретический справочный материал по программированию на Python, варианты индивидуальных заданий к лабораторным работам.

УДК 004.42
ББК 32.973

ЛАБОРАТОРНАЯ РАБОТА №1
Основы языка программирования Python

Цель занятия: приобретение и закрепление теоретических знаний и практических навыков при программировании с типами данных и переменными на Python.

Теоретическая информация:

Введение в Python

Python – это высокоуровневый, интерпретируемый язык программирования с динамической типизацией. Он обладает простым и понятным синтаксисом, что делает его отличным выбором для начинающих программистов.

Типы данных и переменные

В Python существует несколько основных типов данных:

- int (целые числа)
- float (числа с плавающей запятой)
- str (строки)
- bool (логические значения True и False)

Переменные в Python создаются присвоением значения имени.

Пример:

```
x = 10
name = "John"
```

Индивидуальные задания:

1. Создайте переменную age и присвойте ей ваш возраст.
2. Создайте переменную name и присвойте ей ваше имя. Выведите на экран приветствие, используя эту переменную.
3. Создайте переменную pi и присвойте ей значение числа π (пи). Выведите значение на экран.
4. Создайте переменные x и y и присвойте им числовые значения. Перемножьте их и выведите результат.
5. Создайте переменную is_student и присвойте ей логическое значение True или False в зависимости от того, являетесь ли вы студентом.
6. Создайте переменную my_list и присвойте ей список из нескольких чисел.
7. Выведите первый элемент списка my_list.
8. Создайте переменную my_string и присвойте ей строку с вашим любимым цветом.
9. Выведите длину строки из предыдущего задания.

10. Создайте словарь my_dict с данными о вас (имя, возраст, место проживания).

11. Выведите на экран возраст из словаря my_dict.

12. Создайте две переменные, num1 и num2, и присвойте им числовые значения. Поменяйте значения между ними без использования третьей переменной.

13. Создайте переменную user_input и попросите пользователя ввести свой возраст. Преобразуйте введенное значение в целое число.

14. Создайте переменную radius и попросите пользователя ввести радиус круга. Вычислите площадь круга и выведите её на экран.

15. Создайте переменную user_name и попросите пользователя ввести свое имя. Затем выведите его в верхнем регистре.

16. Создайте переменную num3 и присвойте ей значение 10. Увеличьте её значение на 5.

17. Создайте переменную text и присвойте ей строку с вашим любимым цветом. Добавьте к этой строке слово "цвет" и выведите результат.

18. Создайте переменную user_input и попросите пользователя ввести какое-либо число. Проверьте, является ли это число четным, и выведите соответствующее сообщение.

19. Создайте переменную list1 и присвойте ей список строк. Добавьте в этот список ещё одну строку.

20. Создайте переменную tuple1 и присвойте ей кортеж из нескольких чисел. Попробуйте изменить один из элементов кортежа и посмотрите, что произойдет.

ЛАБОРАТОРНАЯ РАБОТА №2
Операторы и выражения

Цель занятия: получение практических навыков работы с операторами, арифметическими выражениями на Python.

Теоретическая информация:

Арифметические операторы

Python поддерживает арифметические операторы, такие как +, -, *, / для выполнения математических операций.

Оператор	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Деление по модулю
//	Целочисленное деление
**	Возведение в степень

Оператор	Пример	Эквивалентная операция
=	a = b	a = b
+=	a += b	a = (a + b)
-=	a -= b	a = (a – b)
*=	a *= b	a = (a * b)
/=	a /= b	a = (a / b)
%=	a %= b	a = (a % b)
//=	a //= b	a = (a // b)
**=	a **= b	a = (a ** b)

Операторы сравнения

Операторы сравнения, такие как ==, !=, <, >, <=, >=, используются для сравнения значений и возвращают булево значение (True или False).

Оператор	Проверяемое условие
= =	Равенство
! =	Неравенство
>	Больше
<	Меньше
> =	Больше либо равно
< =	Меньше либо равно

Оператор	Операция
and	Логическое «И»
or	Логическое «ИЛИ»
not	Логическое «НЕ»

Индивидуальные задания:

1. Напишите программу, которая складывает два числа и выводит результат.
2. Напишите программу, которая вычитает одно число из другого и выводит результат.
3. Напишите программу, которая умножает два числа и выводит результат.
4. Напишите программу, которая делит одно число на другое и выводит результат.
5. Напишите программу, которая находит остаток от деления одного числа на другое и выводит результат.
6. Напишите программу, которая возводит число в квадрат и выводит результат.
7. Напишите программу, которая находит квадратный корень числа и выводит результат.
8. Напишите программу, которая считает площадь прямоугольника, используя длины его сторон.
9. Напишите программу, которая вычисляет площадь треугольника по формуле Герона, используя длины его сторон.
10. Напишите программу, которая конвертирует температуру в градусах Цельсия в градусы Фаренгейта и выводит результат.
11. Напишите программу, которая проверяет, является ли число четным, и выводит соответствующее сообщение.
12. Напишите программу, которая проверяет, делится ли число на 5 и 3 одновременно, и выводит соответствующее сообщение.
13. Напишите программу, которая проверяет, является ли год високосным, и выводит соответствующее сообщение.
14. Напишите программу, которая принимает число от пользователя и определяет, является ли оно положительным, отрицательным или нулем.
15. Напишите программу, которая определяет, является ли число простым, и выводит соответствующее сообщение.
16. Напишите программу, которая принимает число от пользователя и проверяет, является ли оно кратным 3 и/или 5, и выводит соответствующее сообщение.
17. Напишите программу, которая находит среднее арифметическое из трех чисел.
18. Напишите программу, которая принимает два числа и находит наибольший общий делитель (НОД) с использованием алгоритма Евклида.
19. Напишите программу, которая считает сумму цифр введенного числа.
20. Напишите программу, которая принимает число и выводит его зеркальное отражение (например, для числа 12345 результат будет 54321).

ЛАБОРАТОРНАЯ РАБОТА №3
Условные операторы

Цель занятия: получение теоретических знаний и практических навыков работы с условными операторами на Python.

Теоретическая информация:

Условный оператор

Условный оператор if – это одна из фундаментальных конструкций в Python и других языках программирования. Он позволяет выполнять определенный блок кода, только если определенное условие истинно (True). Если условие ложно (False), то выполнится альтернативный блок кода (если он указан с использованием оператора else).

Пример условного оператора if:

```
if условие:
    # Код, который выполняется, если условие истинно
else:
    # Код, который выполняется, если условие ложно
```

Где:

- условие – это выражение, которое Python вычисляет как истинное (True) или ложное (False). Если условие истинно, то код внутри блока if выполняется, в противном случае выполнится код в блоке else, если он есть.
- : - двоеточие используется для обозначения начала блока кода, который будет выполняться в зависимости от условия.
- Блоки кода после if и else должны быть отступлены на один и тот же уровень (обычно с отступом в 4 пробела или одной табуляции).

Пример:

```
x = 15
if x > 10:
    print("x больше 10")
else:
    print("x не больше 10")
```

В этом примере, x > 10 - это условие. Если x больше 10 (что истинно в данном случае), то будет выполнен блок кода после if, и на экран будет выведено "x больше 10". В противном случае будет выполнен блок кода после else, и будет выведено "x не больше 10".

Дополнительные условия

Можно также использовать дополнительные условия с операторами elif (сокращение от "else if") для проверки нескольких альтернативных условий.

Пример:

```
x = 15
if x > 20:
   print("x больше 20")
elif x > 10:
   print("x больше 10, но не больше 20")
else:
   print("x не больше 10")
```

Этот код проверяет три условия последовательно и выводит соответствующее сообщение в зависимости от значения x.

Индивидуальные задания:

1. Напишите программу, которая проверяет, является ли число положительным, отрицательным или нулем.
2. Напишите программу, которая определяет, является ли число четным или нечетным.
3. Напишите программу, которая проверяет, является ли год високосным.
4. Напишите программу, которая определяет, принадлежит ли точка с координатами (x, y) к четвертой четверти координатной плоскости.
5. Напишите программу, которая определяет, принадлежит ли точка с координатами (x, y) к границе круга радиусом R и центром в начале координат.
6. Напишите программу, которая определяет, является ли треугольник с заданными сторонами (a, b, c) прямоугольным.
7. Напишите программу, которая находит максимальное из трех чисел.
8. Напишите программу, которая находит минимальное из четырех чисел.
9. Напишите программу, которая определяет, является ли год високосным и выводит сообщение о том, сколько дней в феврале в этом году.
10. Напишите программу, которая определяет, является ли треугольник с заданными сторонами (a, b, c) равносторонним, равнобедренным или разносторонним.
11. Напишите программу, которая проверяет, является ли введенный символ буквой верхнего регистра (заглавной).
12. Напишите программу, которая проверяет, является ли символ гласной буквой (a, e, i, o, u).
13. Напишите программу, которая определяет, является ли год введенным пользователем числом высокого или низкого столетия (например, 20-й или 21-й).

14. Напишите программу, которая принимает возраст пользователя и выводит сообщение о том, может ли он участвовать в выборах (старше 18 лет).

15. Напишите программу, которая проверяет, является ли введенное число положительным и четным.

16. Напишите программу, которая определяет, является ли введенное число кратным 3 и/или 5.

17. Напишите программу, которая проверяет, введен ли пароль правильно (пароль: "python123").

18. Напишите программу, которая определяет, является ли год введенным пользователем числом столетия (например, 20-й или 21-й) и выводит сообщение о том, является ли год високосным.

19. Напишите программу, которая проверяет, может ли треугольник с заданными сторонами (a, b, c) существовать.

20. Напишите программу, которая принимает оценку пользователя и выводит сообщение о его успеваемости (например, "Отлично", "Хорошо", "Удовлетворительно", "Неудовлетворительно").

ЛАБОРАТОРНАЯ РАБОТА №4
Циклы

Цель занятия: получение теоретических знаний и практических навыков работы с циклами на Python.

Теоретическая информация:

Цикл for

Цикл for используется для выполнения определенного блока кода заданное количество раз или для перебора элементов в последовательности (например, в списке или строке).

Он имеет следующий формат:

```
for переменная in последовательность:
    # Код, выполняющийся в каждой итерации цикла
```

Где:

- переменная – это переменная, которая будет принимать значения из последовательности в каждой итерации.
- последовательность – это объект, который может быть перебран, например, список, строка или результат вызова функции range().

Пример использования цикла for:

```
for i in range(5):
    print(i)
```

В этом примере range(5) создает последовательность чисел от 0 до 4, и цикл for перебирает эти значения и выводит их на экран.

Цикл while

Цикл while выполняет блок кода до тех пор, пока заданное условие истинно (True).

Он имеет следующий формат:

```
while условие:
    # Код, выполняющийся в каждой итерации цикла
```

Где:

- условие – это выражение, которое оценивается как True или False. Цикл будет выполняться, пока условие истинно.

Пример использования цикла while:

```
x = 0
while x < 5:
    print(x)
    x += 1
```

В этом примере, цикл while выполняется, пока x меньше 5. На каждой итерации x выводится на экран, и x увеличивается на 1. Как только x становится равным или больше 5, выполнение цикла прекращается.

Цикл while полезен, когда неизвестно точное количество итераций заранее, и выполнение зависит от условия.

Оба цикла, for и while, полезны для выполнения повторяющихся задач, и выбор между ними зависит от конкретной задачи и требований.

Индивидуальные задания:

1. Напишите программу, которая выводит числа от 1 до 10 с использованием цикла for.
2. Напишите программу, которая выводит все четные числа от 1 до 20 с использованием цикла for.
3. Напишите программу, которая выводит сумму чисел от 1 до 100 с использованием цикла for.
4. Напишите программу, которая выводит таблицу умножения для числа 7 от 1 до 10 с использованием цикла for.
5. Напишите программу, которая запрашивает у пользователя число и выводит все его делители с использованием цикла for.
6. Напишите программу, которая находит факториал числа, введенного пользователем, с использованием цикла for.
7. Напишите программу, которая находит сумму цифр введенного пользователем числа с использованием цикла for.
8. Напишите программу, которая проверяет, является ли введенное пользователем число простым, с использованием цикла for.
9. Напишите программу, которая выводит числа от 10 до 1 с использованием цикла while.
10. Напишите программу, которая выводит все нечетные числа от 1 до 15 с использованием цикла while.
11. Напишите программу, которая запрашивает у пользователя число и выводит таблицу умножения для этого числа от 1 до 10 с использованием цикла while.
12. Напишите программу, которая выводит числа Фибоначчи (последовательность, где каждое число равно сумме двух предыдущих) до 100 с использованием цикла while.
13. Напишите программу, которая принимает от пользователя число и выводит все его делители с использованием цикла while.

14. Напишите программу, которая находит сумму цифр введенного пользователем числа с использованием цикла while.

15. Напишите программу, которая проверяет, является ли введенное пользователем число палиндромом (читается справа налево и слева направо одинаково) с использованием цикла while.

16. Напишите программу, которая находит среднее арифметическое всех чисел от 1 до 50 с использованием цикла for и оператора range().

17. Напишите программу, которая находит сумму всех нечетных чисел от 1 до 99 с использованием цикла for и оператора range().

18. Напишите программу, которая находит наименьший общий делитель (НОК) двух чисел, введенных пользователем, с использованием цикла while.

19. Напишите программу, которая выводит все числа от 1 до 100, но вместо чисел, кратных 3, выводит "Fizz", а вместо чисел, кратных 5, выводит "Buzz", а если число кратно и 3, и 5, то выводит "FizzBuzz".

20. Напишите программу, которая выводит треугольник из звездочек (*) с использованием вложенных циклов. Высоту треугольника вводит пользователь.

ЛАБОРАТОРНАЯ РАБОТА №5
Функции

Цель занятия: получение теоретических знаний и практических навыков при разработке пользовательских функций на Python.

Теоретическая информация:

Создание функций

Функции в Python объявляются с использованием ключевого слова def, за которым следует имя функции, параметры (если они есть) и двоеточие. Затем идет блок кода функции, который будет выполняться, когда функция вызывается.

Формат объявления функции выглядит следующим образом:

```
def имя_функции(параметры):
    # Блок кода функции
```

Пример создания функции:

```
def say_hello():
    print("Привет!")
```

Эта функция say_hello не принимает никаких параметров и просто выводит на экран строку "Привет!".

Вызов функций

Для вызова функции используется ее имя, за которым следуют круглые скобки, в которых могут быть переданы аргументы (значения, которые функция будет использовать внутри себя).

Формат вызова функции выглядит так:

```
имя_функции(аргументы)
```

Пример вызова функции:

```
say_hello()
```

В этом примере вызывается функция say_hello, которая выводит "Привет!" на экран.

Параметры функций

Функции могут принимать аргументы (параметры), которые используются внутри функции для выполнения определенных действий. Параметры объявляются в скобках при определении функции и могут быть использованы внутри блока кода функции.

Пример функции с параметрами:

```
def greet(name):
    print("Привет, " + name + "!")
```

Здесь name – это параметр функции, и его значение можно передать при вызове функции.

Возвращаемые значения функций:

Функции могут возвращать результат своей работы с использованием ключевого слова return. Возвращаемое значение может быть использовано в коде, который вызывает функцию.

Пример функции с возвращаемым значением:

```
def add(x, y):
    result = x + y
    return result
```

В этой функции add два параметра x и y, и она возвращает их сумму.

При вызове функции с возвращаемым значением, результат можно сохранить в переменной:

```
sum_result = add(3, 5)
print(sum_result)  # Выводит 8
```

Эти основы создания и вызова функций в Python позволяют вам создавать модульный и читаемый код, разделяя его на маленькие, логические блоки. Функции также могут использоваться для повторного использования кода и улучшения его структуры.

Индивидуальные задания:

1. Напишите функцию say_hello(), которая выводит на экран приветствие.
2. Напишите функцию add(x, y), которая принимает два аргумента и возвращает их сумму.
3. Напишите функцию subtract(x, y), которая принимает два аргумента и возвращает их разницу.
4. Напишите функцию multiply(x, y), которая принимает два аргумента и возвращает их произведение.
5. Напишите функцию divide(x, y), которая принимает два аргумента и возвращает результат деления первого на второе.
6. Напишите функцию is_even(x), которая принимает число и возвращает True, если оно четное, и False, если нечетное.

7. Напишите функцию is_prime(x), которая принимает число и возвращает True, если оно простое, и False, если составное.

8. Напишите функцию calculate_factorial(x), которая вычисляет факториал числа x и возвращает его значение.

9. Напишите функцию sum_of_digits(x), которая принимает число и возвращает сумму его цифр.

10. Напишите функцию reverse_string(s), которая принимает строку s и возвращает ее в обратном порядке.

11. Напишите функцию find_max(numbers), которая принимает список чисел и возвращает наибольшее из них.

12. Напишите функцию find_min(numbers), которая принимает список чисел и возвращает наименьшее из них.

13. Напишите функцию count_occurrences(lst, target), которая принимает список lst и элемент target, и возвращает количество раз, которое target встречается в списке.

14. Напишите функцию is_palindrome(s), которая принимает строку s и возвращает True, если она является палиндромом (читается справа налево и слева направо одинаково), и False, если нет.

15. Напишите функцию calculate_power(x, n), которая возводит число x в степень n и возвращает результат.

16. Напишите функцию print_pattern(n), которая выводит следующий узор для заданного n:

```
1
12
123
1234
...
```

17. Напишите функцию calculate_gcd(x, y), которая находит наибольший общий делитель (НОД) двух чисел x и y и возвращает его.

18. Напишите функцию calculate_lcm(x, y), которая находит наименьшее общее кратное (НОК) двух чисел x и y и возвращает его.

19. Напишите функцию is_perfect_number(x), которая определяет, является ли число x совершенным числом (сумма всех его делителей, кроме самого числа, равна числу) и возвращает True или False.

20. Напишите функцию calculate_fibonacci(n), которая возвращает список первых n чисел Фибоначчи.

ЛАБОРАТОРНАЯ РАБОТА №6
Списки

Цель занятия: получение теоретических знаний и практических навыков работы со списками на Python.

Теоретическая информация:

Создание списков

Списки – это упорядоченные коллекции элементов, которые могут содержать объекты разных типов данных, такие как числа, строки, другие списки и многое другое. Списки создаются с использованием квадратных скобок [], в которых элементы разделяются запятыми.

Пример создания списка:

```
my_list = [1, 2, 3, 4, 5]
```

В этом примере создается список my_list, который содержит пять целых чисел.

Индексация элементов

Элементы списка нумеруются, начиная с нуля. Вы можете получить доступ к элементам списка, используя индекс в квадратных скобках.

Например:

```
print(my_list[0])  # Выведет 1
```

Здесь my_list[0] обращается к первому элементу списка my_list, который имеет индекс 0.

Отрицательная индексация

Вы также можете использовать отрицательные индексы для обращения к элементам списка с конца. Например, -1 обозначает последний элемент, -2 - предпоследний, и так далее.

Пример:

```
print(my_list[-1])  # Выведет 5 (последний элемент)
print(my_list[-2])  # Выведет 4 (предпоследний элемент)
```

Длина списка

Чтобы узнать количество элементов в списке, вы можете использовать функцию len():

Пример:

```
length = len(my_list)
```

```
print(length)  # Выведет 5 (количество элементов в списке)
```

Изменение элементов списка

Элементы списка можно изменять, присваивая им новые значения по их индексу:

Пример:

```
my_list[1] = 10  # Заменит второй элемент (с индексом 1) на 10
```

Добавление и удаление элементов

Чтобы добавить элемент в конец списка, вы можете использовать метод append():

Пример:

```
my_list.append(6) # Добавит элемент 6 в конец списка
```

Вставка элемента

Для вставки элемента в определенное место списка, вы можете использовать метод insert():

Пример:

```
my_list.insert(2, 7)  # Вставит элемент 7 на позицию с индексом 2
```

Удаление элемента по значению

Чтобы удалить элемент из списка, вы можете использовать метод remove() по значению элемента:

Пример:

```
my_list.remove(3)  # Удалит элемент со значением 3 из списка
```

Удаление элемента по индексу

Для удаления элемента по индексу используется оператор del:

Пример:

```
del my_list[0]  # Удалит первый элемент (с индексом 0)
```

Срезы (slicing)

Для того чтобы получить подсписок из списка с помощью срезов. Срезы позволяют выбрать диапазон элементов с определенным началом, концом и шагом.

Формат среза выглядит следующим образом:

```
подсписок = список [начало:конец:шаг]
```

Примеры срезов:

```
my_list = [1, 2, 3, 4, 5]
subset = my_list[1:4]  # Выберет элементы с индексами 1, 2 и 3
subset2 = my_list[:3]   # Выберет элементы с начала списка до индекса 2 (не включительно)
subset3 = my_list[::2]  # Выберет каждый второй элемент из списка
```

Проверка наличия элемента в списке

Для проверки, содержит ли список определенный элемент, вы можете использовать оператор in:

Пример:

```
if 3 in my_list:
    print("3 содержится в списке")
```

Для обхода элементов списка вы можете использовать цикл for.

Например, чтобы вывести каждый элемент списка на экран:

```
my_list = [1, 2, 3, 4, 5]
for element in my_list:
    print(element)
```

Этот цикл пройдется по каждому элементу списка и выполнит заданное действие (в данном случае, вывод на экран) для каждого элемента.

Функции и методы списков

Списки имеют множество встроенных функций и методов для выполнения различных операций:

append(): Добавляет элемент в конец списка.
insert(): Вставляет элемент на указанную позицию.
remove(): Удаляет элемент из списка по значению.
pop(): Удаляет и возвращает элемент с указанным индексом (по умолчанию – последний).
index(): Возвращает индекс первого вхождения элемента с указанным значением.
count(): Возвращает количество вхождений элемента в список.
sort(): Сортирует элементы списка.
reverse(): Изменяет порядок элементов списка на обратный.

Метод списка	Описание
`list.append(x)`	Добавляет элемент *x* в конец списка
`list.extend(L)`	Добавляет все элементы списка *L* в конец списка
`list.insert(i,x)`	Вставляет элемент *x* в позицию перед индексом *i* в списке
`list.remove(x)`	Удаляет первый элемент *x* из списка
`list.pop(i)`	Удаляет элемент с индексом *i* и возвращает его
`list.index(x)`	Возвращает индекс первого элемента *x* в списке
`list.count(x)`	Возвращает количество вхождений элемента *x* в список
`list.sort()`	Сортирует элементы списка по возрастанию
`list.reverse()`	Обращает порядок следования элементов

Списки – это мощный инструмент для хранения и обработки данных, и они используются очень часто в программировании для работы с коллекциями элементов.

Индивидуальные задания:

1. Создайте пустой список с именем my_list и выведите его на экран.
2. Создайте список чисел от 1 до 10 и выведите его на экран.
3. Создайте список с вашими любимыми цветами и выведите его на экран.
4. Выведите первый элемент списка из предыдущего задания.
5. Выведите последний элемент списка из предыдущего задания.
6. Измените второй элемент списка из предыдущего задания на новый цвет и выведите список снова.
7. Добавьте к списку из предыдущего задания еще один цвет и выведите его.
8. Удалите третий элемент из списка и выведите его.
9. Создайте копию списка из предыдущего задания и выведите ее.
10. Создайте список чисел от 1 до 20 с использованием генератора списка и выведите его.
11. Напишите программу, которая находит сумму всех чисел в списке из предыдущего задания.
12. Напишите программу, которая находит среднее арифметическое всех чисел в списке из предыдущего задания.
13. Напишите программу, которая находит наименьшее и наибольшее число в списке из предыдущего задания и выводит их.
14. Напишите программу, которая находит индекс (позицию) определенного элемента в списке.
15. Напишите программу, которая удаляет все дубликаты из списка и выводит новый список.
16. Напишите программу, которая сортирует список чисел в порядке возрастания и выводит его.
17. Напишите программу, которая объединяет два списка в один и выводит результат.
18. Напишите программу, которая находит пересечение (общие элементы) двух списков и выводит результат.
19. Напишите программу, которая находит разницу между двумя списками (элементы, которые есть в одном списке, но отсутствуют в другом) и выводит результат.
20. Напишите программу, которая определяет, является ли список палиндромом (читается одинаково слева направо и справа налево) и выводит True или False.

ЛАБОРАТОРНАЯ РАБОТА №7
Строки

Цель занятия: получение теоретических знаний и практических навыков работы со строками на Python.

Теоретическая информация:

Создание строк

Строки в Python – это последовательности символов, их можно создавать, заключая текст в одинарные (') или двойные (") кавычки.

Пример создания строки:

```
my_string = "Привет, мир!"
```

Python также позволяет использовать тройные кавычки (''' или """) для создания многострочных строк:

Пример:

```
multi_line_string = """Это
многострочная
строка."""
```

Доступ к символам и подстрокам

Вы можете обращаться к отдельным символам строки, используя индексацию, как в списках. Индексы начинаются с 0 для первого символа.

Пример:

```
my_string = "Привет, мир!"
print(my_string[0])  # Выведет 'П'
```

Чтобы получить подстроку, вы можете использовать срезы (slicing):

Пример:

```
substring = my_string[7:12]  # Получит подстроку "мир!"
```

Длина строки

Чтобы узнать длину строки (количество символов в ней), используйте функцию len():

Пример:

```
length = len(my_string)
print(length)  # Выведет 13 (количество символов в строке)
```

Операции со строками

Python поддерживает различные операции со строками, такие как конкатенация (соединение строк), умножение на число и другие.

Например:

```
str1 = "Hello, "
str2 = "world!"
concatenated_string = str1 + str2  # Конкатенация строк
print(concatenated_string)  # Выведет "Hello, world!"
repeated_string = str1 * 3  # Умножение строки на число
print(repeated_string)  # Выведет "Hello, Hello, Hello, "
```

Методы строк

Строки имеют множество методов для выполнения различных операций.

Например:

upper(): Преобразует строку в верхний регистр.

lower(): Преобразует строку в нижний регистр.

strip(): Удаляет пробелы и символы новой строки в начале и конце строки.

split(): Разделяет строку на подстроки по заданному разделителю и возвращает список.

replace(): Заменяет одну подстроку другой.

find(): Ищет подстроку в строке и возвращает индекс первого вхождения (или -1, если не найдено).

Примеры использования некоторых методов строк:

```
my_string = "   Привет, мир!   "
trimmed_string = my_string.strip()  # Удаление пробелов по краям
print(trimmed_string)  # Выведет "Привет, мир!"
words = my_string.split(",")  # Разделение строки по запятой
print(words)  # Выведет список ["   Привет", " мир!   "]
replaced_string = my_string.replace("мир", "вселенная")  # Замена "мир" на "вселенная"
print(replaced_string)  # Выведет "   Привет, вселенная!   "
```

Эти методы помогают вам выполнять разнообразные операции со строками в Python. Строки являются важной частью программирования и используются для хранения и обработки текстовых данных.

В программах на языке Python для работы с переменными строкового типа используются различные операторы, представленные в таблице ниже.

Оператор	Описание	Пример
+	Конкатенация (объединение) строк	'Hello' + 'Mike'
*	Повторение строки указанное число раз	'Hello' * 2
[ ]	Выбор символа по указанному индексу	'Hello' [0]
[:]	Извлечение среза по указанному диапазону индексов	'Hello' [0 : 4]
in	Проверка вхождения — возвращает True, если символ или подстрока присутствует в строке	'H' in 'Hello'
not in	Обратная операция — возвращает True, если символ или подстрока в строке отсутствует	'h' not in 'Hello'
r/R	«Сырая строка» — подавление экранирующей последовательности	print(r'\n')
''' '''	Строка документации — для описания модуля, функции, класса или метода	def sum(a,b) : ''' Add Args '''

Индивидуальные задания:

1. Создайте строку с вашим именем и выведите ее на экран.
2. Создайте строку, состоящую из нескольких предложений, и выведите ее на экран.
3. Выведите длину строки из предыдущего задания.
4. Выведите первый символ строки из предыдущего задания.
5. Выведите последний символ строки из предыдущего задания.
6. Создайте новую строку, объединив две строки из заданий 1 и 2, и выведите ее.
7. Создайте строку, содержащую ваше имя в верхнем регистре, и выведите ее.
8. Создайте строку, содержащую ваше имя в нижнем регистре, и выведите ее.
9. Создайте строку с вашим полным именем и разделите ее на отдельные слова, используя метод split().
10. Создайте строку и замените в ней одно слово на другое, используя метод replace().
11. Создайте строку, которая состоит из нескольких слов, разделенных пробелами, и посчитайте, сколько раз слово встречается в этой строке.
12. Создайте строку, содержащую числа от 1 до 10, разделенные запятыми, и разделите ее на список чисел, используя метод split().

13. Создайте строку и удалите из нее все пробелы, используя метод replace().

14. Создайте строку и проверьте, начинается ли она с определенного префикса, используя метод startswith().

15. Создайте строку и проверьте, заканчивается ли она определенным суффиксом, используя метод endswith().

16. Создайте строку и проверьте, является ли она числом, используя методы isdigit() или isnumeric().

17. Создайте строку и преобразуйте ее в верхний регистр, используя метод upper().

18. Создайте строку и преобразуйте ее в нижний регистр, используя метод lower().

19. Создайте строку и удалите лишние пробелы в начале и конце строки, используя метод strip().

20. Создайте строку и разделите ее на список символов, используя метод list().

ЛАБОРАТОРНАЯ РАБОТА №8
Словари

Цель занятия: получение теоретических знаний и практических навыков работы с условными операторами на Python.

Теоретическая информация:

Создание словарей

Словари в Python - это коллекции, которые хранят пары ключ-значение. Каждый элемент словаря имеет уникальный ключ, и ключи обычно используются для доступа к соответствующему значению. Словари создаются с использованием фигурных скобок {} или функции dict(), и элементы в них записываются в формате "ключ": "значение".

Пример создания словаря:

```
my_dict = {"имя": "Джон", "возраст": 30, "город": "Нью-Йорк"}
```

В этом примере создается словарь my_dict с тремя парами ключ-значение.

Обращение к элементам словаря

Элементы словаря можно получить, обратившись к ним по ключу в квадратных скобках []:

Пример:

```
name = my_dict["имя"]  # Получит значение по ключу "имя"
print(name)  # Выведет "Джон"
```

Если ключа нет в словаре, будет сгенерировано исключение KeyError. Чтобы избежать этого, можно использовать метод get(), который возвращает значение по ключу или значение по умолчанию, если ключ отсутствует:

Пример:

```
name = my_dict.get("имя", "Неизвестно")  # Получит значение по ключу "имя" или "Неизвестно", если ключ отсутствует
print(name)  # Выведет "Джон"
```

Изменение и добавление элементов

Вы можете изменять значения элементов словаря, обратившись к ним по ключу и присвоив новое значение:

Пример:

```
my_dict["возраст"] = 31  # Изменит значение по ключу "возраст"
```

Чтобы добавить новый элемент в словарь, просто присвойте новому ключу значение:

Пример:
my_dict["email"] = "john@example.com" # Добавит новую пару ключ-значение

Удаление элементов
Чтобы удалить элемент из словаря, используйте ключевое слово del и указание ключа:

Пример:
del my_dict["город"] # Удалит элемент с ключом "город"

Проверка наличия ключа
Для проверки наличия ключа в словаре используйте оператор in:

Пример:
```
if "возраст" in my_dict:
    print("Ключ 'возраст' существует в словаре")
```

Получение ключей и значений
Вы можете получить список всех ключей и значений в словаре с использованием методов keys() и values():

Пример:
```
keys_list = my_dict.keys()      # Возвращает список ключей
values_list = my_dict.values()  # Возвращает список значений
```

Итерация по словарю
Для итерации по словарю и получения ключей и значений можно использовать цикл for:

Пример:
```
for key in my_dict:
    value = my_dict[key]
    print(key, value)
```

Или можно использовать метод items(), который возвращает пары ключ-значение:

```
for key, value in my_dict.items():
    print(key, value)
```

Словари являются удобными для хранения и организации данных, особенно когда каждому элементу нужно присвоить уникальный идентификатор (ключ). Они широко используются в Python для хранения информации и создания структурированных данных.

Индивидуальные задания:

1. Создайте пустой словарь с именем my_dict и выведите его на экран.
2. Создайте словарь, представляющий информацию о вас (имя, возраст, место проживания) и выведите его на экран.
3. Выведите значение по ключу "имя" из словаря из предыдущего задания.
4. Измените значение возраста в словаре из задания 2 и выведите словарь снова.
5. Добавьте в словарь из задания 2 новую пару ключ-значение, представляющую вашу профессию, и выведите словарь.
6. Удалите ключ "место проживания" из словаря и выведите словарь без этого ключа.
7. Проверьте, существует ли ключ "возраст" в словаре из задания 2, и выведите True или False.
8. Создайте список словарей, представляющий информацию о нескольких людях, и выведите его на экран.
9. Выведите информацию о конкретном человеке из списка словарей из предыдущего задания.
10. Измените значение в одном из словарей из списка и выведите обновленный список.
11. Создайте словарь, представляющий информацию о товаре (название, цена, количество) и выведите его на экран.
12. Увеличьте количество товара в словаре из задания 11 и выведите обновленный словарь.
13. Удалите товар из словаря из задания 11 и выведите словарь без этого товара.
14. Создайте словарь, представляющий информацию о студенте (имя, оценки по предметам) и выведите его на экран.
15. Добавьте оценку по новому предмету в словарь из задания 14 и выведите обновленный словарь.
16. Посчитайте средний балл студента из словаря из задания 14 и выведите его.
17. Создайте словарь, представляющий информацию о книге (название, автор, год выпуска) и выведите его на экран.
18. Измените год выпуска книги в словаре из задания 17 и выведите обновленный словарь.
19. Создайте словарь, представляющий информацию о путешествии (место, даты, бюджет) и выведите его на экран.
20. Измените бюджет для путешествия в словаре из задания 19 и выведите обновленный словарь.

ЛАБОРАТОРНАЯ РАБОТА №9
Обработка ошибок

Цель занятия: получение теоретических знаний и практических навыков обработки ошибок при разработки кодов на Python.

Теоретическая информация:

Обработка исключений в Python позволяет управлять ошибками, которые могут возникнуть во время выполнения программы, чтобы код не завершался аварийно. Это особенно полезно при работе с потенциально проблемными операциями или данными.

Конструкция try...except

Конструкция try...except позволяет вам попытаться выполнить определенный блок кода, и если в нем возникнет исключение (ошибка), вы можете перехватить это исключение и выполнить альтернативный блок кода. Формат try...except выглядит следующим образом:

Пример:

```
try:
    # Блок кода, в котором может возникнуть исключение
except Исключение as переменная:
    # Блок кода, который выполняется при возникновении исключения
```

Где:

- Исключение – это тип исключения, которое вы хотите перехватить. Это может быть стандартным типом исключения (например, ZeroDivisionError, TypeError, FileNotFoundError, и так далее) или пользовательским типом исключения.
- переменная (необязательно) - это имя переменной, в которой будет сохранен объект исключения для дальнейшего анализа.

Пример использования try...except:

```
try:
    x = 10 / 0
except ZeroDivisionError as e:
    print("Произошло деление на ноль:", e)
```

В этом примере код попытается выполнить деление на ноль, что вызовет исключение ZeroDivisionError. Затем этот тип исключения будет перехвачен блоком except, и в переменной e будет сохранен объект исключения. Вы можете использовать этот объект для вывода информации об ошибке.

Блоки except

Вы можете иметь несколько блоков except, чтобы обрабатывать разные типы исключений по-разному.

Например:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Деление на ноль!")
except ValueError:
    print("Ошибка значения!")
```

Если возникнет исключение ZeroDivisionError, будет выполнен первый блок except, а если возникнет исключение ValueError, будет выполнен второй блок except.

Блок finally

Вы также можете использовать блок finally, который будет выполнен в любом случае, даже если исключение было перехвачено или не было перехвачено. Это может быть полезно для освобождения ресурсов или выполнения завершающих операций.

Пример:

```
try:
    # Блок кода
except Исключение as переменная:
    # Обработка исключения
finally:
    # Блок, который выполняется всегда
```

Обработка исключений позволяет гибко управлять ошибками и обеспечивать более надежную работу ваших программ. Она помогает избежать аварийных завершений программы и предоставляет возможность обработать ошибку и продолжить выполнение программы в зависимости от ситуации.

Индивидуальные задания:

1. Напишите программу, которая делит одно число на другое, и обработайте исключение деления на ноль.
2. Напишите программу, которая преобразует строку в целое число, и обработайте исключение, если в строке содержится не число.
3. Напишите программу, которая открывает файл для чтения и обработайте исключение, если файл не существует.

4. Напишите программу, которая пытается преобразовать строку в число с плавающей запятой и обработайте исключение, если строка не может быть преобразована.

5. Напишите программу, которая пытается открыть файл для записи и обработайте исключение, если файл заблокирован другой программой.

6. Напишите программу, которая пытается получить доступ к элементу списка по индексу и обработайте исключение, если индекс выходит за пределы списка.

7. Напишите программу, которая запрашивает у пользователя число и делит 10 на это число, обработайте исключение, если пользователь ввел ноль.

8. Напишите программу, которая просит пользователя ввести целое число, и обработайте исключение, если пользователь ввел не число.

9. Напишите программу, которая пытается преобразовать строку в целое число и обработайте исключение, если строка не может быть преобразована.

10. Напишите программу, которая открывает файл и пытается прочитать его содержимое, обработайте исключение, если файл не существует.

11. Напишите программу, которая пытается выполнить деление двух чисел и обработайте исключение, если второе число равно нулю.

12. Напишите программу, которая просит пользователя ввести целое число и возводит его в степень, обработайте исключение, если пользователь ввел не целое число.

13. Напишите программу, которая пытается открыть файл для записи и обработайте исключение, если нет разрешения на запись.

14. Напишите программу, которая пытается получить доступ к ключу в словаре и обработайте исключение, если ключ отсутствует.

15. Напишите программу, которая пытается разделить два числа, и обработайте исключение, если одно из чисел не является числом.

16. Напишите программу, которая открывает файл и пытается прочитать его содержимое, обработайте исключение, если файл имеет неверный формат.

17. Напишите программу, которая просит пользователя ввести число и находит его квадратный корень, обработайте исключение, если число отрицательное.

18. Напишите программу, которая пытается преобразовать строку в число с плавающей запятой и обработайте исключение, если строка имеет неверный формат.

19. Напишите программу, которая открывает файл и пытается прочитать его содержимое, обработайте исключение, если файл слишком большой.

20. Напишите программу, которая пытается выполнить математическую операцию (например, деление) и обработайте исключение, если операция невозможна.

Список рекомендуемой литературы

1. Гэддис Т. Начинаем программировать на Python. – 4-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 768 с.
2. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2019. – 161 с. – (Бакалавр. Прикладной курс). – ISBN 978-5-534-10971-9. – Текст: электронный // ЭБС Юрайт [сайт]. – URL: https://urait.ru/bcode/437489
3. Шелудько, В. М. Основы программирования на языке высокого уровня Python: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 146 с. – ISBN 978-5-9275-2649-9. – Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. – URL: http://www.iprbookshop.ru/87461.html
4. Шелудько, В. М. Язык программирования высокого уровня Python. Функции, структуры данных, дополнительные модули: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 107 с. – ISBN 978-5-9275-2648-2. – Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. – URL: http://www.iprbookshop.ru/87530.html

БЕЖАНОВА Елена Хусиновна
ДЖАТДОЕВ Алим Сеит-Алиевич.

ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

Лабораторных практикум для обучающихся на 1 курса направление подготовки 01.03.02 Прикладная математика и информатика очной формы обучения

Корректор Чагова О.Х.
Редактор Чагова О.Х.

Сдано в набор 16.01.2024 г.
Формат 60x84/16
Бумага офсетная.
Печать офсетная.
Усл. печ. л.1,86
Заказ № 4850
Тираж 100 экз.

Оригинал-макет подготовлен
в Библиотечно-издательском центре СКГА
369000, г. Черкесск, ул. Ставропольская, 36