Утверждаю
Директор
должность руководителя ОУ
Государственного бюджетного профессионального образовательного учреждения Иркутской области «Братский политехнический колледж» (ГБПОУ ИО «БрПК»)
наименование образовательного учреждения (в соответствии с уставом ОУ)
_____ / А.Э. Ишкова
Личная подпись *А. Э. Ишкова*
«02»ноября 2020 г.
М.П.

ДОПОЛНИТЕЛЬНАЯ ПРОФЕССИОНАЛЬНАЯ ПРОГРАММА

Государственного бюджетного образовательного учреждения среднего профессионального образования Иркутской области "Братский политехнический колледж"
(ГБОУ СПО БрПК)
наименование образовательного учреждения

по программе повышения квалификации

Основы программирования на Python

СОДЕРЖАНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дополнительной профессиональной программе
по программе повышения квалификации
«Основы программирования на Python»

Программа повышения квалификации **«Основы программирования на Python»** представляет собой комплект документов, разработанных и утвержденных Государственным бюджетным профессиональным образовательным учреждением Иркутской области «Братский политехнический колледж» с учетом потребностей регионального рынка труда, требований федеральных органов исполнительной власти и соответствующих отраслевых требований, на основе профессионального стандарта «Разработчик Web и мультимедийных приложений» утвержденного приказом Министерства труда и социальной защиты Российской Федерации от 18.01.2017 г.,№ 44н, зарегистрировано Министерством юстиции 31.01. 2017 г., № 45481. В результате анализа функциональной карты вида профессиональной деятельности была определена трудовая функция, результат работы представлен в таблице 1.

Таблица 1.- Связь дополнительной профессиональной программы с профессиональным стандартам

Наименование программы	Наименование выбранного профессионального стандарта, трудовая функция.	Уровень квалификации трудовой функции
«Основы программирования на Python»	«Разработчик Web и мультимедийных приложений» **Трудовая функция:** Кодирование на языках web-программирования.	4

Программа регламентирует цели, ожидаемые результаты, содержание, условия и технологии реализации образовательного процесса, оценку качества подготовки слушателя по данной программе и включает в себя: учебный план, программу курса, календарный учебный график, а также оценочные и методические материалы, обеспечивающие качество подготовки обучающихся.

1.1. Нормативно-правовая основа разработки дополнительной профессиональной программы.

Нормативно-правовую базу программы составляют:

- Закон об образовании в Российской Федерации от 29.12.2012 г. № 273-ФЗ;

- Порядок организации и осуществления образовательной деятельности по дополнительным профессиональным программам, утвержденный Приказом Министерства образования и науки Российской Федерации от 1 июля 2013 г. № 499, зарегистрированного Министерством юстиции Российской Федерации 20 августа 2013 г. Регистрационный № 29444 (В ред. Приказа Минобрнауки России от 15.11.2013 № 1244);

- Квалификационный справочник должностей руководителей, специалистов и других служащих, утвержденный Постановлением Минтруда РФ от 21 августа 1998 г. N 37 (в ред. Постановлений Минтруда России от 21.01.2000 N 7, от 04.08.2000 N 57, от 20.04.2001 N 35,от 31.05.2002 N 38, от 20.06.2002 N 44,от 28.07.2003 N 59, от 12.11.2003 N 75, Приказов Минздравсоцразвития России от 25.07.2005 N 461,от 07.11.2006 N 749, от 17.09.2007 N 605,от 29.04.2008 N 200, от 14.03.2011 N 194, Приказов Минтруда России от 15.05.2013 N 205,от 12.02.2014 N 96)

- Постановление Правительства Российской Федерации от 15 августа 2013 г. № 706 «Об утверждении правил оказания платных образовательных услуг

- Приказ Минтруда России от 12 апреля 2013 г. № 148н «Об утверждении уровней квалификаций в целях разработки проектов профессиональных стандартов»

- Приказ Минтруда России от 29 апреля 2013 г. № 170-н «Об утверждении методических рекомендаций по разработке профессионального стандарта»

- Методические рекомендации-разъяснения по разработке дополнительных профессиональных программ на основе профессиональных стандартов, письмо Министерства образования и науки Российской Федерации от 22 апреля 2015 г. № ВК-1032/06.

- Устав образовательного учреждения.

Учебный план составлен в соответствии с «Порядок организации и осуществления образовательной деятельности по дополнительным профессиональным программам», утвержденный Приказ Министерства образования и науки Российской Федерации от 1 июля 2013 г. № 499, и современными требованиями, обусловленными необходимостью перехода на качественно новый

уровень подготовки специалистов, обеспечивающих их мобильность, социальную защищенность, конкурентоспособность на рынке труда города Братска и региона.

Программа предназначена для категории лиц, достигших возраста 16 лет.

К освоению дополнительных профессиональных программ допускаются:

1) лица, имеющие среднее профессиональное и (или) высшее образование;

2) лица, получающие среднее профессиональное и (или) высшее образование.

При освоении дополнительной профессиональной программы параллельно с получением среднего профессионального образования и (или) высшего образования удостоверение о повышении квалификации и выдаются одновременно с получением соответствующего документа об образовании и о квалификации.

Продолжительность обучения согласно календарному графику обучения. В ходе освоения программы слушатели приобретают знания и практические навыки разработки программ на языке программирования Python.

1.2. Цель реализации программы. Планируемые результаты обучения

Цель и планируемые результаты обучения:

Цель программы направлена на освоение следующих профессиональных компетенций:

Профессиональные компетенции	Практический опыт	Умения	Знания
1	2	3	4
Создание программного кода в соответствии с техническим заданием (готовыми спецификациями) Оптимизация программного кода с использованием специализированных программных средств Написание программного кода с использованием языков про-	Создание программного кода в соответствии с техническим заданием (готовыми спецификациями) Оптимизация программного кода с использованием специализированных программных средств Написание программного кода с использованием	Применять выбранные языки программирования для написания программного кода Использовать выбранную среду программирования и средства системы управления базами данных Использовать возможности имеющейся программной архитектуры ИР	Синтаксис выбранного языка программирования, особенности программирования на этом языке Особенности выбранной среды программирования и системы управления базами данных

граммирования, определения и манипулирования данными	языков программирования, определения и манипулирования данными		Стандартные библиотеки выбранного языка программирования Компоненты программно-технических архитектур ИР, существующие приложения и интерфейсы взаимодействия с ними

1.3. Организационно-педагогические условия, формы аттестации*:*

Программа обеспечена учебно-методической документацией и материалами по всем учебным курсам. Педагогические сотрудники формируют и хранят в кабинетах и лабораториях учебно-методические комплексы по каждому учебному курсу.

Колледж обеспечивает возможность доступа студентов к новой учебной и методической литературе по информационным курсам в читальном зале библиотеки.

В колледже обеспечена возможность выхода в информационные сети через Интернет. Для реализации программы учебного курса в колледже оборудованы 11 компьютерных классов.

Продолжительность занятий – 45 минут. Занятия проводятся парами. Между уроками пары перерыв 5 минут. Между парами перерыв 10 минут.

Контроль и оценка результатов освоения учебного курса, осуществляется преподавателями в процессе проведения устных опросов, практических занятий, лабораторных занятий. Текущий контроль по учебным курсам проводится в пределах учебного времени, как традиционными, так и инновационными методами, включая информационные технологии. Система оценок – пятибалльная. При освоении учебных курсов предусматривается проведение промежуточной аттестации в форме практической работы.

После завершения освоения программы итоговая аттестация проводится в форме дифференцированного зачета. Дифференцированный зачет проводится в виде выполнения практической работы на языке программирования Python. Результатом выполнения зачетной работы является работоспособный программный продукт.

1.4. Учебный план

по дополнительной профессиональной программе,
(программа повышения квалификации)
«Основы программирования на Python»

Индекс	Наименование модуля	Форма промежуточной аттестации*[1]	Учебная нагрузка обучающихся (час.)			
			максимальная	самостоятельная учебная работа	Обязательная аудиторная	
					всего занятий	в т. ч. лаб. и практ. занятий
1	**2**		**3**	**4**	**5**	**6**
1	Основы языка Python	Практическая работа	70	36	34	30
2	**Итоговая аттестация** Дифференцированный зачет	Практическая работа	2		2	2
Всего			**72**	**36**	**36**	**32**

[1] **Промежуточная аттестация проводиться за счет времени учебного курса**

1.5. Календарный учебный график по дополнительной профессиональной программе

«Основы программирования на Python»

	Наименование модуля	Виды учебной нагрузки	Недели									Всего часов
			1	2	3	4	5	6	7	8	9	
1	Основы языка Python	обяз.уч	4	4	4	4	4	4	4	4	2	34
		сам.р.с.	4	4	4	4	4	4	4	4	4	36
2	Дифференцированный зачет	Итоговая аттестация									2	2
Всего часов в неделю самостоятельной работы студентов			4	4	4	4	4	4	4	4	4	36
Всего часов в неделю			8	8	8	8	8	8	8	8	8	72

РАБОЧАЯ ПРОГРАММА УЧЕБНОГО МОДУЛЯ

«Основы языка Python»

Организация-разработчик: Государственное бюджетное образовательное учреждение среднего профессионального образования Братский политехнический колледж

Разработчики:

Котова Е.Н., педагог дополнительного образования ГБПОУ ИО «Братский политехнический колледж».

Рассмотрена и одобрена на заседании предметно-цикловой комиссии протокол № 05 от 10. 06. 2020 г., председатель ПЦК Котова Е.Н..

СОДЕРЖАНИЕ

1. ПАСПОРТ ПРОГРАММЫ УЧЕБНОГО МОДУЛЯ «Создание приложений в среде MIT App inventor»

1.1. Область применения программы

Программа модуля «Основы языка Python» предназначена для лиц, желающих получить систему практических знаний в области программирования на языке Python. Новый опыт получат, и начинающие работники в сфере информационных технологий и те, кто уже ведет работу в области разработки программных продуктов.

1.2. Цели и задачи модуля – требования к результатам освоения учебного модуля:

Модуль направлен на освоение трудовой функции:

1. Написание программного кода с использованием языков программирования, определения и манипулирования данными

В результате изучения учебного модуля слушатель должен освоить следующие компетенции:

1. Создание программного кода в соответствии с техническим заданием (готовыми спецификациями).
2. Оптимизация программного кода с использованием специализированных программных средств.

Модуль направлен на приобретение следующего практического опыта:

1. Создание программного кода в соответствии с техническим заданием (готовыми спецификациями).
2. Оптимизация программного кода с использованием специализированных программных средств.

В результате изучения учебного курса слушатель должен:

Уметь:

1. Применять выбранные языки программирования для написания программного кода.
2. Использовать возможности имеющейся программной архитектуры ИР.

Знать:

1. Синтаксис выбранного языка программирования, особенности программирования на этом языке.

2. Стандартные библиотеки выбранного языка программирования.

3. Компоненты программно-технических архитектур ИР, существующие приложения и интерфейсы взаимодействия с ними.

1.3. количество часов на освоение программы учебного модуля:

максимальной учебной нагрузки обучающегося 72 часа, в том числе: обязательной аудиторной учебной нагрузки обучающегося 34 часа; самостоятельной работы обучающегося 36 часов.

2. СТРУКТУРА И СОДЕРЖАНИЕ УЧЕБНОГО КУРСА

2.1. Объем учебного курса и виды учебной работы

Вид учебной работы	***Объем часов***
Максимальная учебная нагрузка (всего)	**72**
Обязательная аудиторная учебная нагрузка (всего)	**34**
в том числе:	
практические занятия	**30**
Самостоятельная работа обучающегося (всего)	**36**
Дифференцированный зачет	**2**

2.2. Тематический план и содержание учебного модуля «Основы языка Python»

Наименование разделов и тем	Содержание учебного материала, лабораторные работы и практические занятия, самостоятельная работа обучающихся		Объем часов
1	**2**		**3**
Тема 1 Введение. Знакомство с Python.	**Содержание учебного материала**		
	1	Язык программирования Python. Знакомство и первая работа в среде разработки Wing.	1
	2	Вычисления и переменные	1
	3	**Практические работы** Первая программа «Hello world!»	2
	Самостоятельная работа Решение задач		4
Тема 2 Построение программы на языке Python.	**Содержание учебного материала**		
	1	Работа со строками и списками. Функции и методы строк. Функции и методы списков	1
	2	Операторы сравнения. Операторы присваивания. Логические операторы	1
	3	**Практические работы** Арифметические операции со строками Сообщение пользователю: написание письма с использованием строк Написание программы расчета количества часов в году	6
	Самостоятельная работа Решение задач		8
Тема 3 Ветвление	1	**Практические работы** Написание программ с использованием инструкции if – elif – else для определения результата	2
	Самостоятельная работа Решение задач		4
Тема 4 Цикл в языке программирования Python	1	**Практические работы** Решение задач с циклом for Решение задач с циклом while	4
	Самостоятельная работа Решение задач		4
Тема 5 Коллекции	1	**Практические работы** Работа с кортежем – turpl Работа со словарем – dict Работа со множествами	6
	Самостоятельная работа Решение задач		6
Тема 6 Функции	1	**Практические работы** Применение и написание функции def Применение рекурсии	6
	Самостоятельная работа Решение задач		6

Тема 7 Работа с файлами	1	**Практические работы** Инструкция from. Создание своего модуля на Python	4
	Самостоятельная работа Решение задач		4
	Дифференцированный зачет		2
Всего			**72**

3. УСЛОВИЯ РЕАЛИЗАЦИИ УЧЕБНОЙ УЧЕБНОГО МОДУЛЯ

3.1 Реализация учебного модуля требует наличия кабинета 201
Оборудование учебного кабинета:
Рабочие места по количеству обучающихся;
Рабочее место преподавателя;
Наглядные пособия (схемы, плакаты, и т.д.);
Технические средства обучения:
Компьютер с лицензионным программным обеспечением;
Телевизор;
Видеопроектор;
Экран;
Локальная сеть;
Выход в глобальную сеть;
Мультимедиапроектор.

3.2. Информационное обеспечение обучения
Перечень рекомендуемых учебных изданий, интернет-ресурсов, дополнительной литературы
Основные источники:
1. К. Ю. Поляков, Е. А. Еремин. Информатика. Углублённый уровень. Учебник для 10 класса в 2 частях. М.: БИНОМ. Лаборатория знаний, 2014.
2. М. Лутц. Изучаем Python. СПб.: Символ-Плюс, 2011.
3. Задачи по программированию. Под ред. С. М. Окулова, М.: БИНОМ. Лаборатория знаний, 2006.
4. С. М. Окулов. Основы программирования. М.: Бином. Лаборатория знаний, 2012.
Дополнительные источники:
1. М. Лутц. Изучаем Python. СПб.: Символ-Плюс, 2011.
2. Информатика и ИКТ. Задачник-практикум в 2 частях. Под ред. И. Г. Семакина и Е. К. Хеннера. М.: БИНОМ. Лаборатория знаний, 2014.
Профильные периодические издания:
1. Webdix.Journal (он-лайн версия) (http://webdix.livejournal.com/)
Специализированные порталы:
1. http://www.intuit.ru
2. http://habrahabr.ru/blogs/programming/
3. Сайт pythontutor.ru — «Питонтьютор».

4. КОНТРОЛЬ И ОЦЕНКА РЕЗУЛЬТАТОВ ОСВОЕНИЯ УЧЕБНОГО МОДУЛЯ

Результаты обучения (освоенные умения, усвоенные знания)	**Формы и методы контроля и оценки результатов обучения**
1	*2*
Умения	
-Применять выбранные языки программирования для написания программного кода -Использовать выбранную среду программирования и средства системы управления базами данных -Использовать возможности имеющейся программной архитектуры ИР	Наблюдение за выполнением практических работ
Знания	
-Синтаксис выбранного языка программирования, особенности программирования на этом языке -Особенности выбранной среды программирования и системы управления базами данных -Стандартные библиотеки выбранного языка программирования -Компоненты программно-технических архитектур ИР, существующие приложения и интерфейсы взаимодействия с ними	Текущий контроль в форме: - ответы на вопросы; Доклад, самостоятельная работа, практические работы

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ИРКУТСКОЙ ОБЛАСТИ
БРАТСКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Комплект оценочных материалов для проведения итоговой аттестации по учебному модулю «Основы языка Python»

I. Паспорт комплекта оценочных материалов

1. Область применения комплекта оценочных материалов.

Комплект оценочных материалов предназначен для оценки результатов освоения учебного модуля **«Основы языка Python»**

Таблица 1

Результаты освоения (объекты оценивания)	Основные показатели оценки результата и их критерии	Тип задания; № задания	Форма аттестации (в соответствии с учебным планом)

ПК 1. Создание программного кода в соответствии с техническим заданием (готовыми спецификациями). ПК 2. Оптимизация программного кода с использованием специализированных программных средств. ПК 3. Написание программного кода с использованием языков программирования, определения и манипулирования данными. **Уметь:** Применять выбранные языки программирования для написания программного кода -Использовать выбранную среду программирования и средства системы управления базами данных -Использовать возможности имеющейся программной архитектуры ИР. **Знать:** -Синтаксис выбранного языка программирования, особенности программирования на этом языке -Особенности выбранной среды программирования и системы управления базами данных -Стандартные библиотеки выбранного языка программирования - Компоненты программно-технических архитектур ИР, существующие приложения и интерфейсы взаимодействия с ними	Создание программного кода в соответствии с техническим заданием (готовыми спецификациями) -Оптимизация программного кода с использованием специализированных программных средств -Написание программного кода с использованием языков программирования, определения и манипулирования данными -Размещение программного кода в страницах, созданных при верстке ИР -Размещение программного кода в клиентской части ИР -Размещение программного кода в серверной части ИР -Оценка и согласование сроков выполнения поставленных задач	Практические задания	Практическая работа

2. Комплект оценочных материалов для промежуточной аттестации

2.1. Задания для проведения промежуточной аттестации

Задание 1

Написать программы согласно условиям задач.

1. Пусть N – произвольное натуральное число ($1 \le N \le 100000$). Предложить алгоритм, который выводит на печать все делители числа N.

2. Некоторые числа, кратные числу 7, при делении на 2, на 3, на 4, на 5 и на 6 дают остаток 1. Предложить алгоритм, который находит наименьшее из таких чисел.

3. Пусть n и p – произвольные натуральные числа ($1 \le n \le 1000$ и $1 \le p \le 1000$). Предложить алгоритм, который находит наименьший общий делитель этих двух чисел.

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ИРКУТСКОЙ ОБЛАСТИ
БРАТСКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Комплект оценочных материалов
для проведения итоговой аттестации
по программе повышения квалификации
«Основы языка Python»

I. Паспорт комплекта оценочных материалов

1. Область применения комплекта оценочных материалов.

Комплект оценочных материалов предназначен для оценки результатов освоения дополнительной профессиональной программы по программе повышения квалификации

«Основы программирования на Python»

Таблица 1

Результаты освоения (объекты оценивания)	Основные показатели оценки результата и их критерии	Тип задания; № задания	Форма аттестации (в соответствии с учебным планом)
ПК 1. Создание программного кода в соответствии с техническим заданием (готовыми спецификациями). ПК 2. Оптимизация программного кода с использованием специализированных программных средств. ПК 3. Написание программного кода с использованием языков программирования, определения и манипулирования данными. **Уметь:** Применять выбранные языки программирования для написания программного кода -Использовать выбранную среду программирования и средства системы управления базами данных -Использовать возможности имеющейся программной архитектуры ИР. **Знать:** -Синтаксис выбранного языка программирования, особенности программирования на этом языке -Особенности выбранной среды программирования и системы управления базами данных -Стандартные библиотеки выбранного языка программирования - Компоненты программно-технических архитектур ИР, существующие приложения и интерфейсы взаимодействия с ними	Создание программного кода в соответствии с техническим заданием (готовыми спецификациями) -Оптимизация программного кода с использованием специализированных программных средств -Написание программного кода с использованием языков программирования, определения и манипулирования данными -Размещение программного кода в страницах, созданных при верстке ИР -Размещение программного кода в клиентской части ИР -Размещение программного кода в серверной части ИР -Оценка и согласование сроков выполнения поставленных задач	Практические задания	Дифференцированный зачет

2. Комплект оценочных материалов для итоговой аттестации

2.1. Задания для проведения итоговой аттестации

ЗАДАНИЕ №1

Программист Вася добирается до работы на общественном транспорте. Дорога занимает около часа, но недавно Вася решил по утрам ходить в бассейн, а значит ему нужно выходить из дома на час раньше. Вася задумался, где он может сэкономить время и пришел к выводу, что было бы не плохо, добираться на работу на машине. Таким образом он сможет сэкономить около 30 минут. Для покупки машины, у Васи не хватает некоторой суммы и поэтому он решил взять автокредит.

В банке Васе предложили взять кредит с аннуитетным графиком. Подсчет процентов по такому кредиту ведется в 2 этапа.

1. Определяется размер ежемесячного платежа (x) по следующей формуле:

$$x = S \times \left(P + \frac{P}{(1+P)^N - 1}\right)$$

Здесь S — сумма займа, P — 1/100 доля процентной ставки (в месяц), N — срок кредитования (в месяцах).

2. Вычисляется доля процентов (I) в ежемесячном взносе по формуле:

$$I = S \times P$$

Здесь S — остаточный объем средств, P — упомянутая ранее процентная ставка. Разберем на примере. Вы планируете взять 200 000 руб. под 12% годовых сроком на 24 месяца. Чтобы вычислить значение P, разделите размер ставки на 100 и затем на 12:

$$12/100/12 = 0{,}01$$

Далее нужно рассчитать размер аннуитетного ежемесячного платежа (по формуле 1). Он получился равным примерно 9 415 руб.

$$x = 200\,000 \times \left(0{,}01 + \frac{0{,}01}{(1+0{,}01)^{24} - 1}\right) \approx 9\,415$$

Затем нужно рассчитать ежемесячные процентные и долговые части в составе платежей по аналогии с таблицей:

Месяцы	Остаток долга	Платеж	Процентная часть	Долговая часть	Остаток долга на конец периода
Первый	200 000	9 415	200 000*0,01=2 000	9 415-2 000=7415	200 000-7 415=192 585
Второй	192 585	9 415	192 585*0,01=1 926	9 415-1 926=7489	192 585-7 489=185 096
Третий	185 096	9 415	185 096*0,01=1 851	9 415-1 851=7564	185 096-7 564=177 532

Помогите Васе определиться на какую сумму он может взять кредит. Напишите программу, которая по сумме кредита, срока кредита и процентной ставке, рассчитает для Васи ежемесячный платеж.

Формат ввода данных

В первой строке вводится сумма кредита в формате «[Сумма кредита:] сумма кредита [руб.]».

Во второй строке вводится срок выплат кредита в формате «[Срок выплат:] срок выплат [год, лет]».

В третей строке вводится процентная ставка в формате «[Процентная ставка:] значение процентной ставки [годовых]».

Формат вывода

Результаты расчетов должны быть представлены за каждый месяц, в виде таблицы: Номер платежа, Сумма платежа, Проценты, Погашено, Остаток.

Пример оформления ввода данных и результатов работы программы.

Ваши данные:

_ _

Сумма кредита: 100 000 руб.

Срок выплат: 1 год

Процентная ставка: 12.9% годовых

_ _

Номер платежа	Сумма платежа	Проценты	Погашено	Остаток
1	8 927 руб.	1 075	7 852	92 148
2	8 927 руб.	990,59	7 936,41	84 211,59
3	8 927 руб.	905,27	8 021,73	76 189,87
4	8 927 руб.	819,04	8 107,96	68 081,91
5	8 927 руб.	731,88	8 195,12	59 886,79
6	8 927 руб.	643,78	8 283,22	51 603,57
7	8 927 руб.	554,74	8 372,26	43 231,31
8	8 927 руб.	464,74	8 462,26	34 769,05
9	8 927 руб.	373,77	8 553,23	26 215,81
10	8 927 руб.	281,82	8 645,18	17 570,63
11	8 927 руб.	188,88	8 738,12	8 832,52
12	8 927 руб.	94,95	8 832,05	0,47
ИТОГО		7 124	100 000	107 124

Государственное бюджетное профессиональное
образовательное учреждение
Иркутской области
«Братский политехнический колледж»

Методическое пособие для обучающихся

Практикум по программирования на языке Python

Братск, 2019

Котова Е.Н., Практикум по программирования на языке Python – Братск, 2019. – 39 с.

Практикум разработан для изучения основ программирования на языке программирования Python.

Одобрено предметно-цикловой комиссией информатики и вычислительной техники протокол № 05 от «10» июня 2020г.

Лабораторная работа. Построение графиков.

Запись алгебраических выражений

Фундаментальная задача программирования — вычисление математических и, в частности, алгебраических функций. Казалось бы, что проще? Однако, запись выражения на языке математики не принимается напрямую языком программирования. Выражение нужно написать в виде, который будет понятен тому или иному языку программирования.

Например, $y = x^2$, должно быть записано как y = x*x или y = x**2.

Упражнение №1

Запишите выражение, заданное формулой, в виде, подходящем для языка Python.

$$y = log_{1+x^2} \frac{e^{\frac{1}{sinx+1}}}{\frac{5}{4} + \frac{1}{x^{15}}}$$

и найдите его значения в точках 1, 10, 1000.

Для вычисления математических функций мы не будем использовать стандартную библиотеку **math**. Т.к. она не работает с векторами. В нашем случае разумней обратить внимание на библиотеку **numpy**. Данная библиотека обеспечивает удобную работу с векторам.

Т.е., если у нас есть вектор x=[1, 2, 3, 4] и мы вызовим numpy.log(x), то логарифм будет взят от каждого элемента списка и возвращен будет список значений.

Аналогичная функция в модуля math ожидает число, т.е. нельзя сделать math.log(x), нужно делать math.log(x[0]) и т.д.

Традиционно библиотека numpy подключается командой:

```
import numpy as np
```

Данный вызов сообщает, что подключить numpy под псевдонимом np. Это делается, чтобы не писать каждый раз:

```
numpy.cos(x)
```

А писать:

```
np.cos(x)
```

Такой код, с более коротким именем библиотеки, элементарно, проще читать.

Основные математические функции и константы функии, которые нам понадобятся из numpy:

Функция библиотеки math	**Математическая функция**
np.pi	Число pi

Функция библиотеки math	Математическая функция
np.e	Число e
np.cos	Косинус
np.sin	Синус
np.tan	Тангенс
np.acos	Арккосинус
np.asin	Арксинус
np.atan	Арктангенс
np.exp	Экспонента
np.log	Логарифм

Функция log вычисляет натуральный логарифм. Чтобы вычислить логарифм по другому основанию, нужно воспользоваться формулой перехода. Например, если мы хотим получить логарифм x по основанию 2, нужно написать:

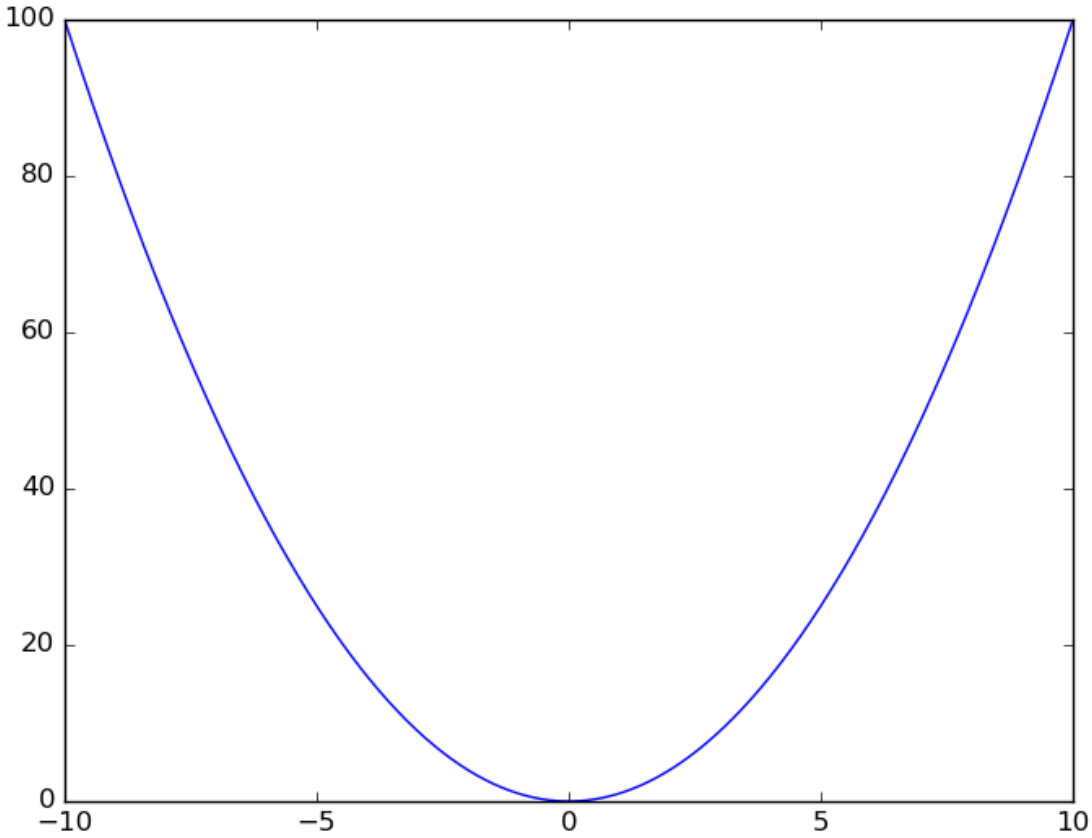
```
np.log(x) / np.log(2)
```

Построение графиков

matplotlib - набор дополнительных модулей (библиотек) языка Python. Предоставляет средства для построения самых разнообразных 2D графиков и диаграмм данных. Отличается простотой использования — для построения весьма сложных и красочно оформленных диаграмм достаточно нескольких строк кода. При этом качество получаемых изображений более чем достаточно для их публикования. Также позволяет сохранять результаты в различных форматах, например Postscript, и, соответственно, вставлять изображения в документы TeX. Предоставляет API для встраивания своих графических объектов в приложения пользователя.

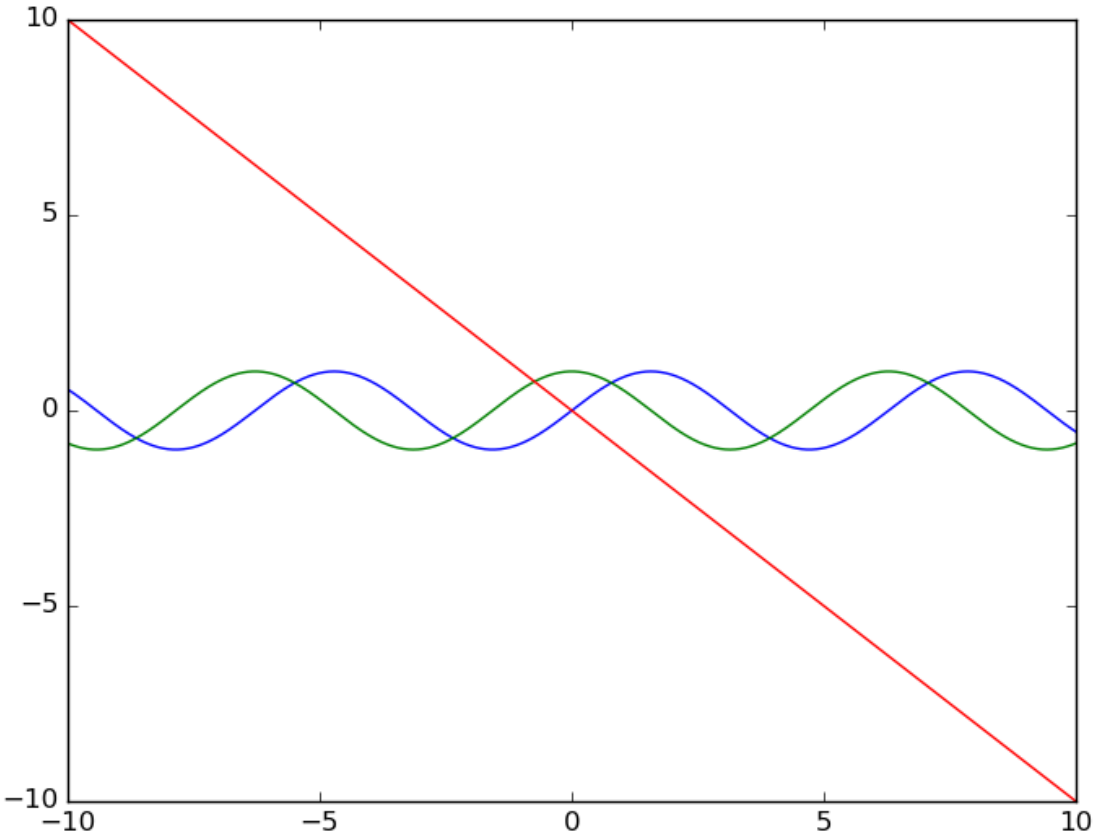
Пример построения графика функции:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-10, 10.01, 0.01)
plt.plot(x, x**2)
plt.show()
```



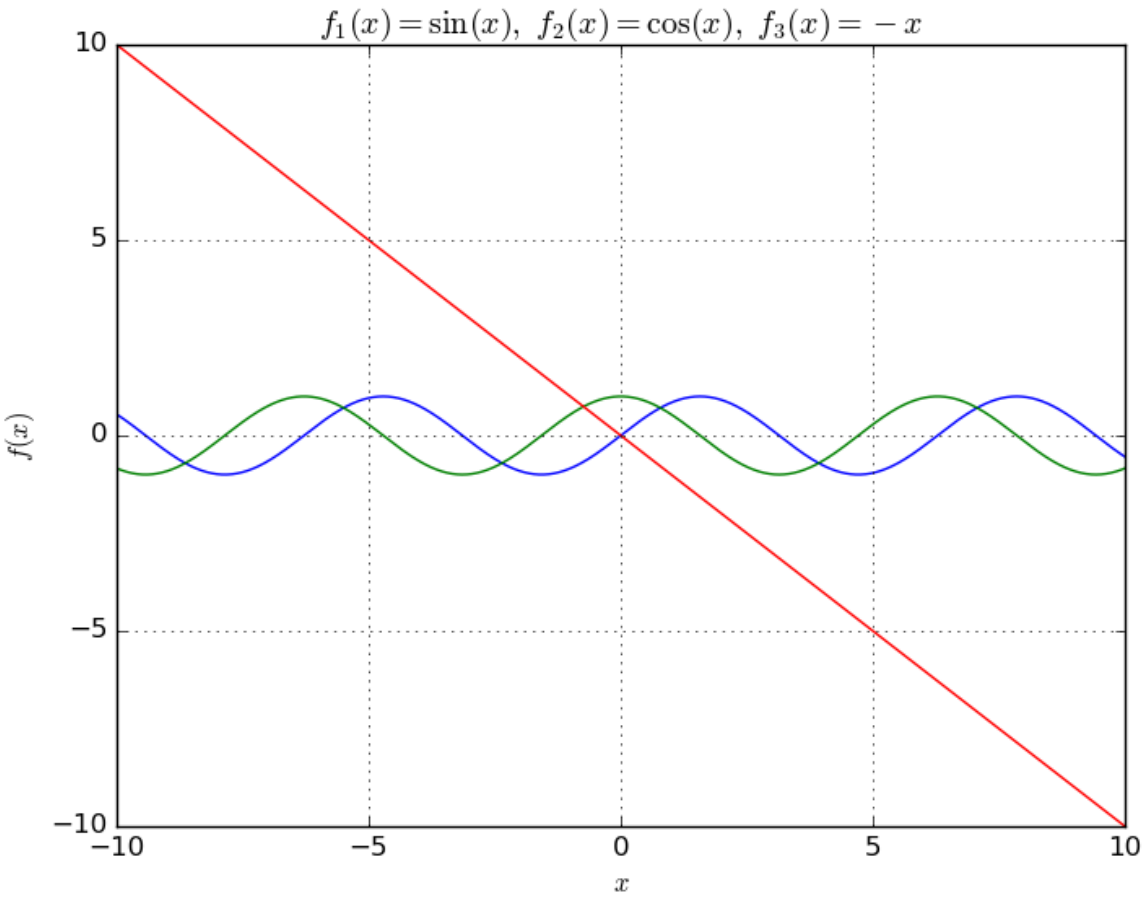
На одном рисунке можно построить несколько графиков функций:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-10, 10.01, 0.01)
plt.plot(x, np.sin(x), x, np.cos(x), x, -x)
plt.show()
```



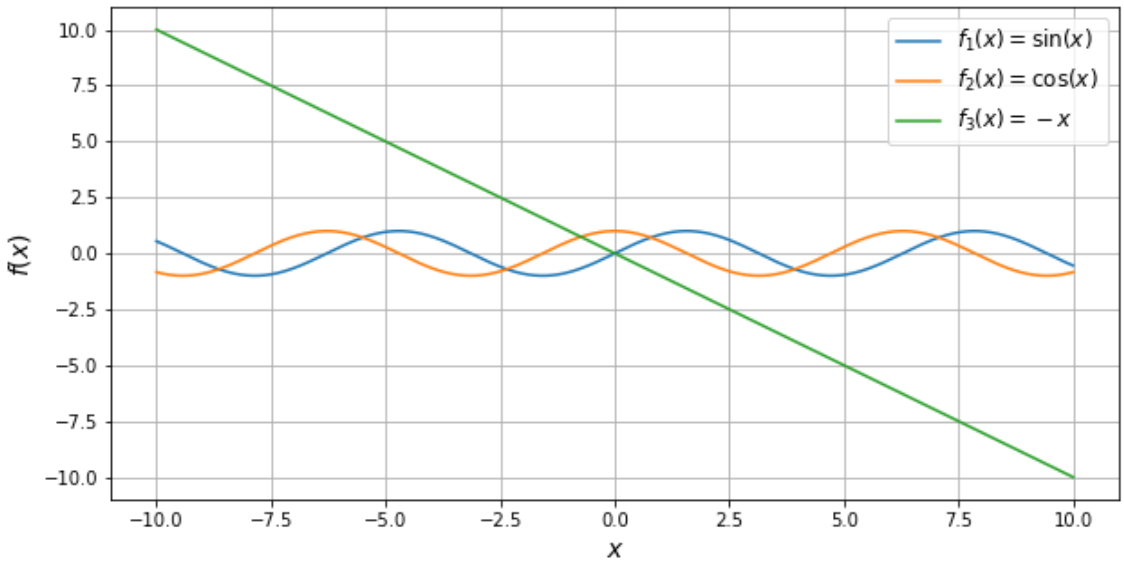
Также довольно просто на график добавить служебную информацию и отобразить сетку:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-10, 10.01, 0.01)
plt.plot(x, np.sin(x), x, np.cos(x), x, -x)
plt.xlabel(r'$x$')
plt.ylabel(r'$f(x)$')
plt.title(r'$f_1(x)=\sin(x),\ f_2(x)=\cos(x),\ f_3(x)=-x$')
plt.grid(True)
plt.show()
```



Или используя legend(), где можно указать место расположения подписей к кривым на графике в параметре loc. Подписи могут быть явно переданы legend((line1, line2, line3), ('label1', 'label2', 'label3')) или могут быть переданы в аргумет label, как в примере ниже. Чтобы сохранить график нужно воспользоваться savefig(figure_name), где figure_name является строкой назания файла с указанием расширения. Для текстовых полей можно изменять шрифт (fontsize), для большей читаемости графика, а его размер указывается с помощью figure(figsize=(10, 5)).

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-10, 10.01, 0.01)
plt.figure(figsize=(10, 5))
plt.plot(x, np.sin(x), label=r'$f_1(x)=\sin(x)$')
plt.plot(x, np.cos(x), label=r'$f_2(x)=\cos(x)$')
plt.plot(x, -x, label=r'$f_3(x)=-x$')
plt.xlabel(r'$x$', fontsize=14)
plt.ylabel(r'$f(x)$', fontsize=14)
plt.grid(True)
plt.legend(loc='best', fontsize=12)
plt.savefig('figure_with_legend.png')
plt.show()
```



Текстовые поля в matplotlib могут содержать разметку **LaTeX**, заключенную в знаки $. Буква r перед кавычками говорит python, что символ "\" следует оставить как есть и не интерпретировать как начало спецсимвола (например, перевода строки - "\n").

Работа с matplotlib основана на использовании графических окон и осей (оси позволяют задать некоторую графическую область). Все построения применяются к текущим осям. Это позволяет изображать несколько графиков в одном графическом окне. По умолчанию создаётся одно графическое окно figure(1) и одна графическая область subplot(111) в этом окне. Команда subplot позволяет разбить графическое окно на несколько областей. Она имеет три параметра: nr, nc, np. Параметры nr и nc определяют количество строк и столбцов на которые разбивается графическая область, параметр np определяет номер текущей области (np принимает значения от 1 до nr*nc). Если nr*nc<10, то передавать параметры nr, nc, np можно без использования запятой. Например, допустимы формы subplot(2,2,1) и subplot(221).

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-10, 10.01, 0.01)
t = np.arange(-10, 11, 1)

#subplot 1
sp = plt.subplot(221)
plt.plot(x, np.sin(x))
plt.title(r'$\sin(x)$')
plt.grid(True)

#subplot 2
sp = plt.subplot(222)
plt.plot(x, np.cos(x), 'g')
plt.axis('equal')
plt.grid(True)
plt.title(r'$\cos(x)$')
```

```
#subplot 3
sp = plt.subplot(223)
plt.plot(x, x**2, t, t**2, 'ro')
plt.title(r'$x^2$')

#subplot 4
sp = plt.subplot(224)
plt.plot(x, x)
sp.spines['left'].set_position('center')
sp.spines['bottom'].set_position('center')
plt.title(r'$x$')

plt.show()
```

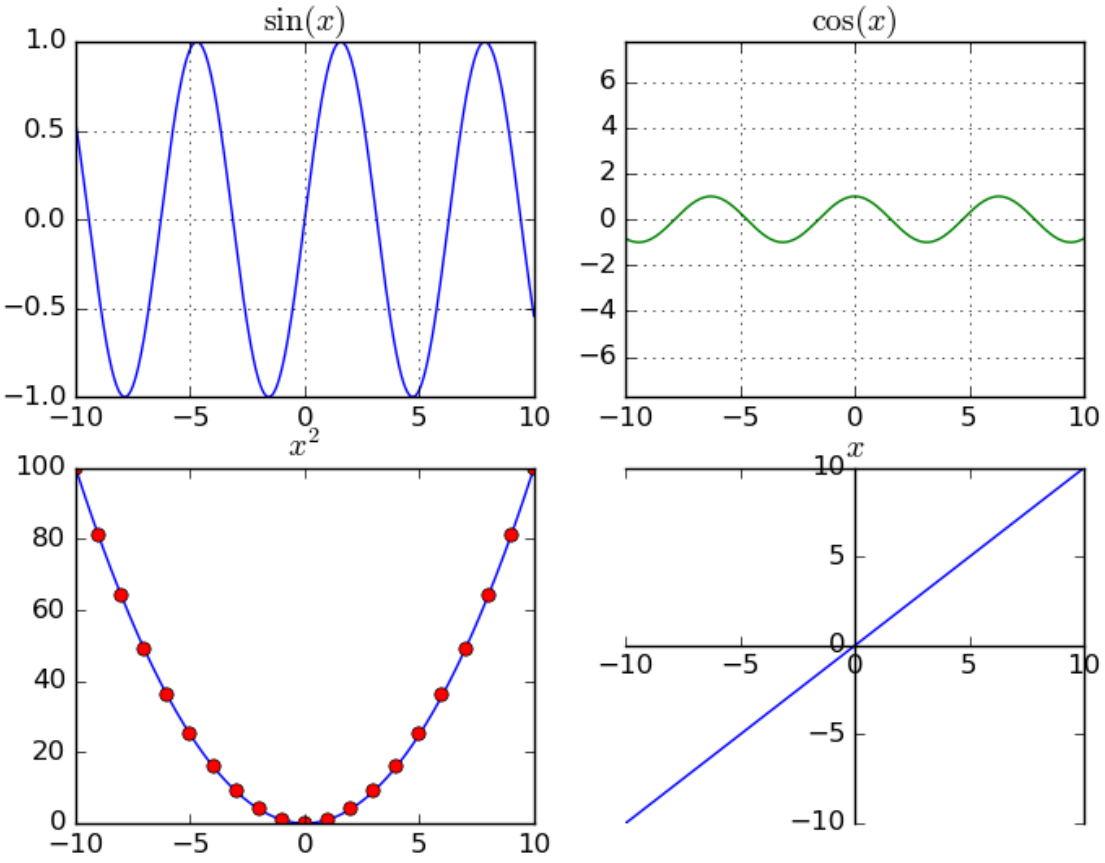
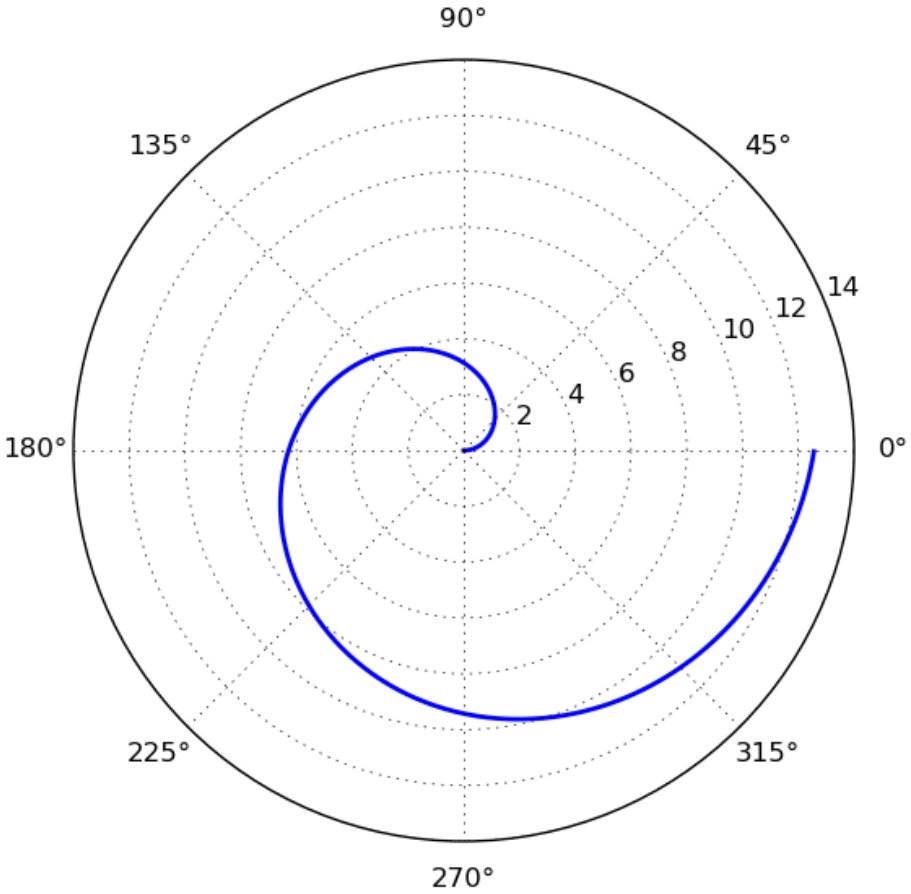


График может быть построен в полярной системе координат, для этого при создании subplot необходимо указать параметр polar=True:

```
import numpy as np
import matplotlib.pyplot as plt
plt.subplot(111, polar=True)
phi = np.arange(0, 2*np.pi, 0.01)
rho = 2*phi
plt.plot(phi, rho, lw=2)
plt.show()
```



Или может быть задан в параметрической форме (для этого не требуется никаких дополнительных действий, поскольку два массива, которые передаются в функцию plot воспринимаются просто как списки координат точек, из которых состоит график):

```
import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0, 2*np.pi, 0.01)
r = 4
plt.plot(r*np.sin(t), r*np.cos(t), lw=3)
plt.axis('equal')
plt.show()
```

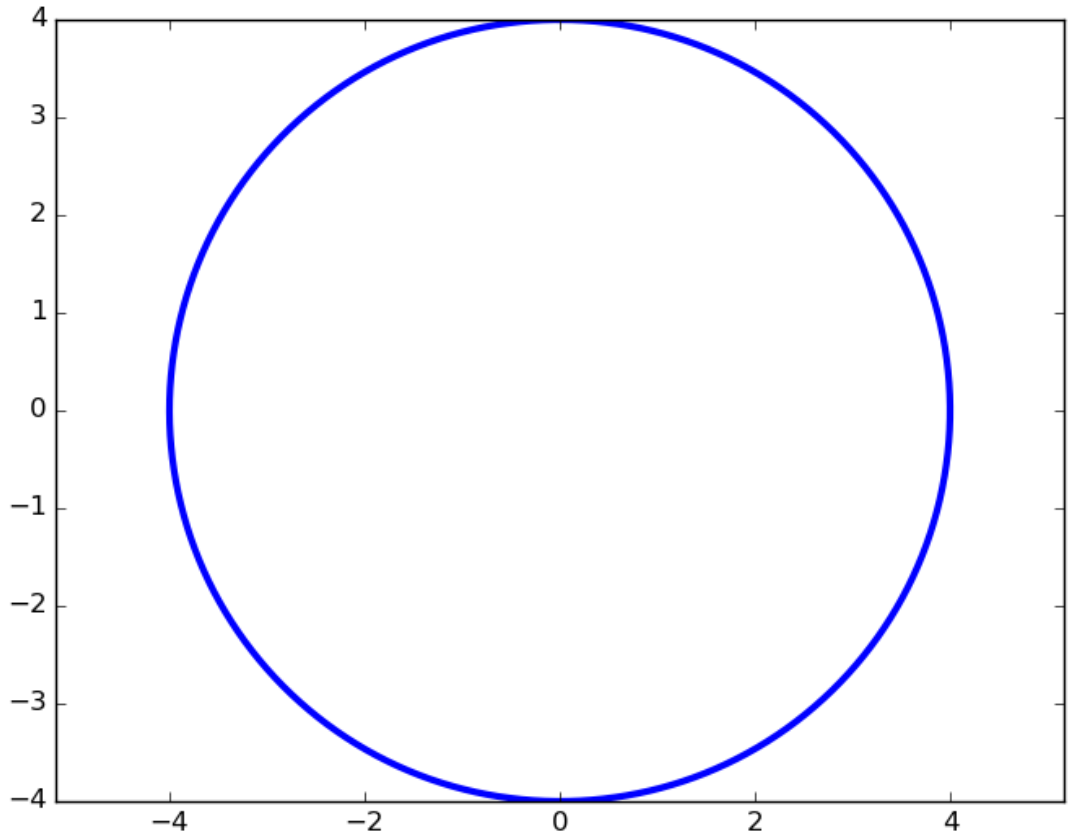
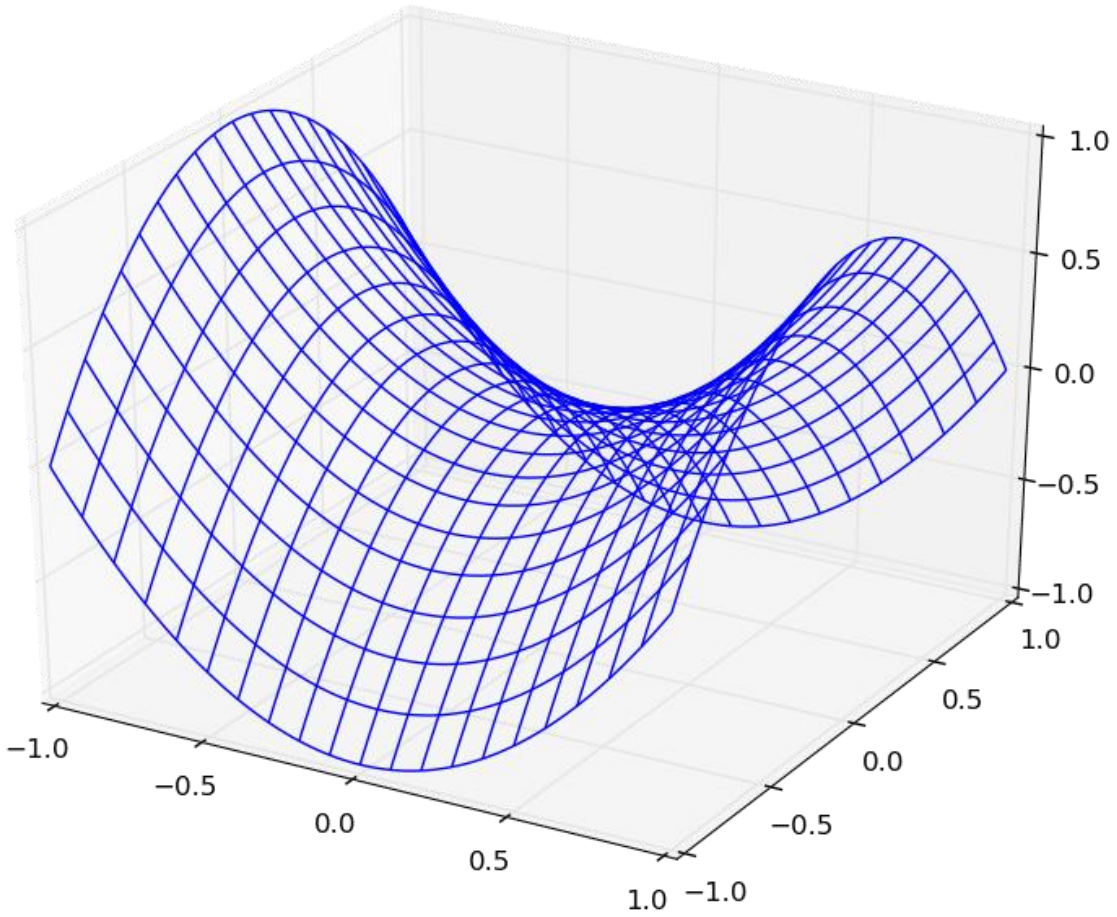


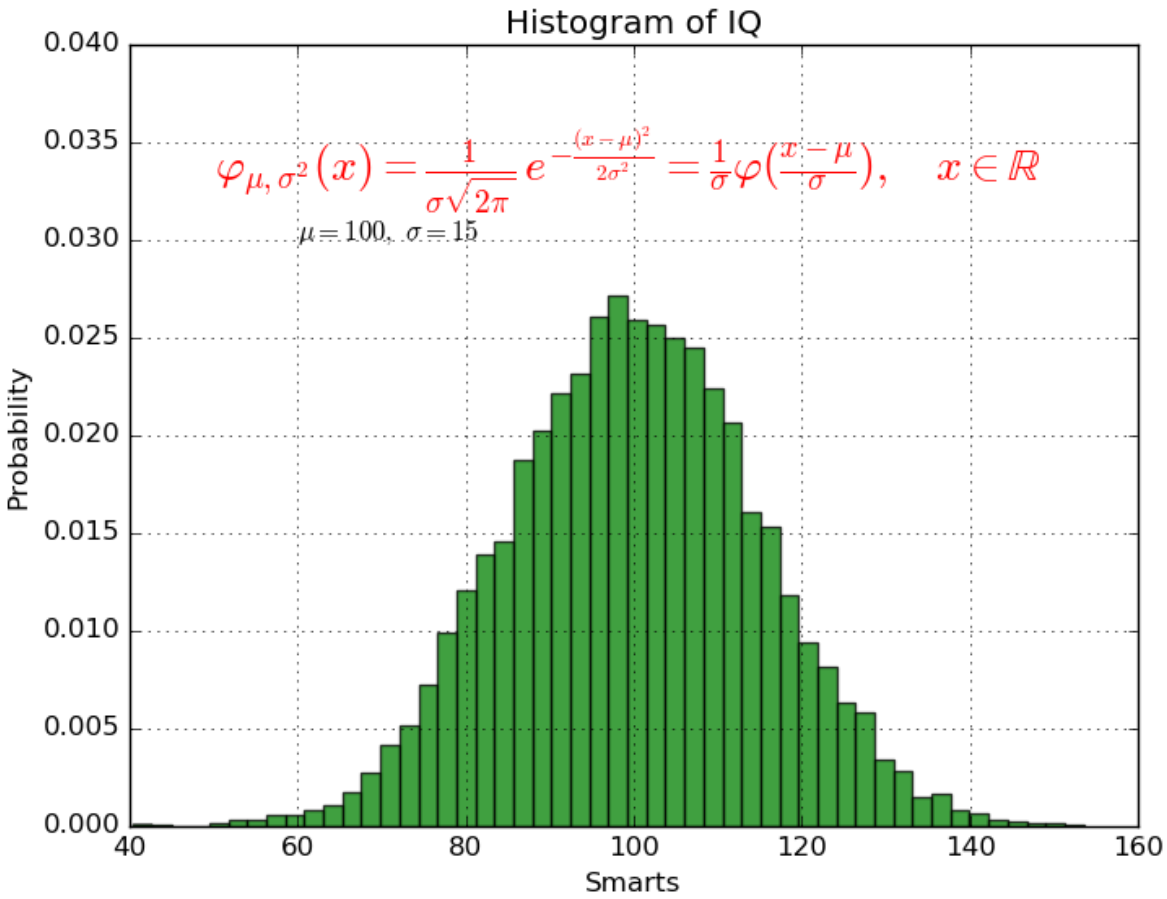
График функции двух переменных может быть построен, например, так:

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
ax = axes3d.Axes3D(plt.figure())
i = np.arange(-1, 1, 0.01)
X, Y = np.meshgrid(i, i)
Z = X**2 - Y**2
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
plt.show()
```



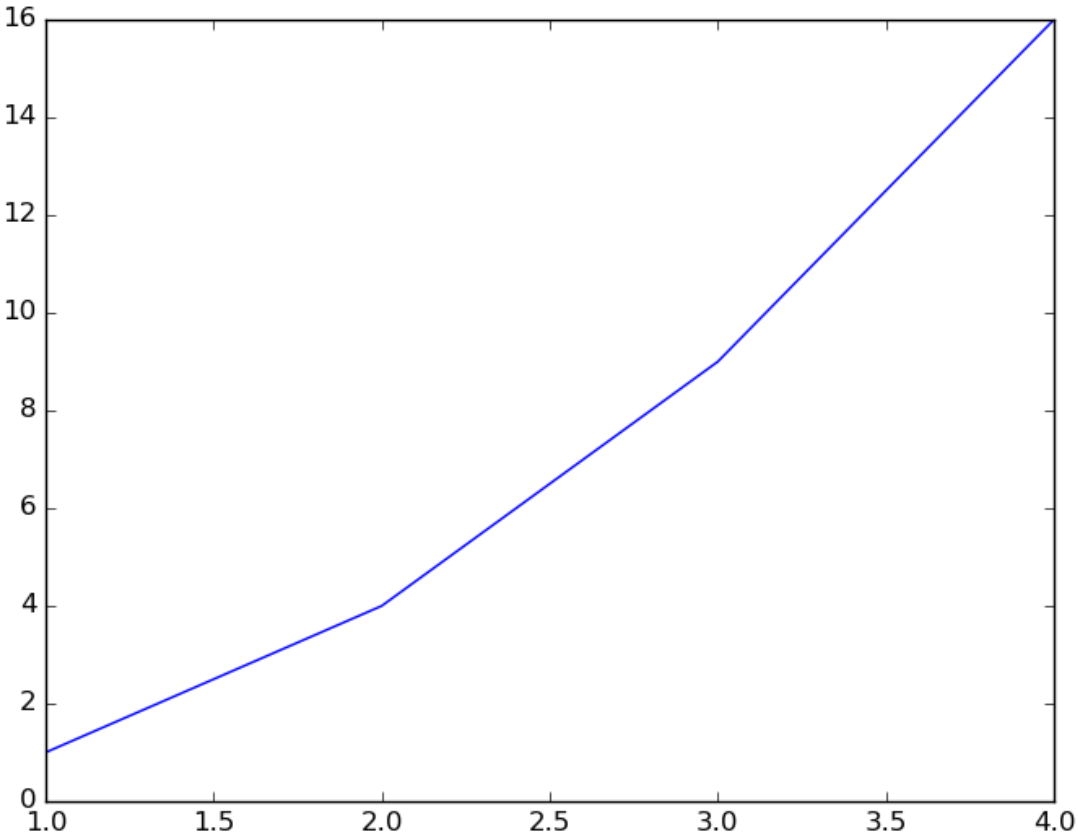
Добавление текста на график: Команду text() можно использовать для добавления текста в произвольном месте (по умолчанию координаты задаются в координатах активных осей), а команды xlabel(), ylabel() и title() служат соответственно для подписи оси абсцисс, оси ординат и всего графика. Для более полной информации смотрите **«Text introduction»** раздел на оф. сайте.

```
import numpy as np
import matplotlib.pyplot as plt
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=True, facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .030, r'$\mu=100,\ \sigma=15$')
plt.text(50, .033, r'$\varphi_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} \,e^{ -\frac{(x- \mu)^2}{2\sigma^2}} = \frac{1}{\sigma} \varphi\left(\frac{x - \mu}{\sigma}\right),\quad x\in\mathbb{R}$', fontsize=20, color='red')
plt.axis([40, 160, 0, 0.04])
plt.grid(True)
plt.show()
```



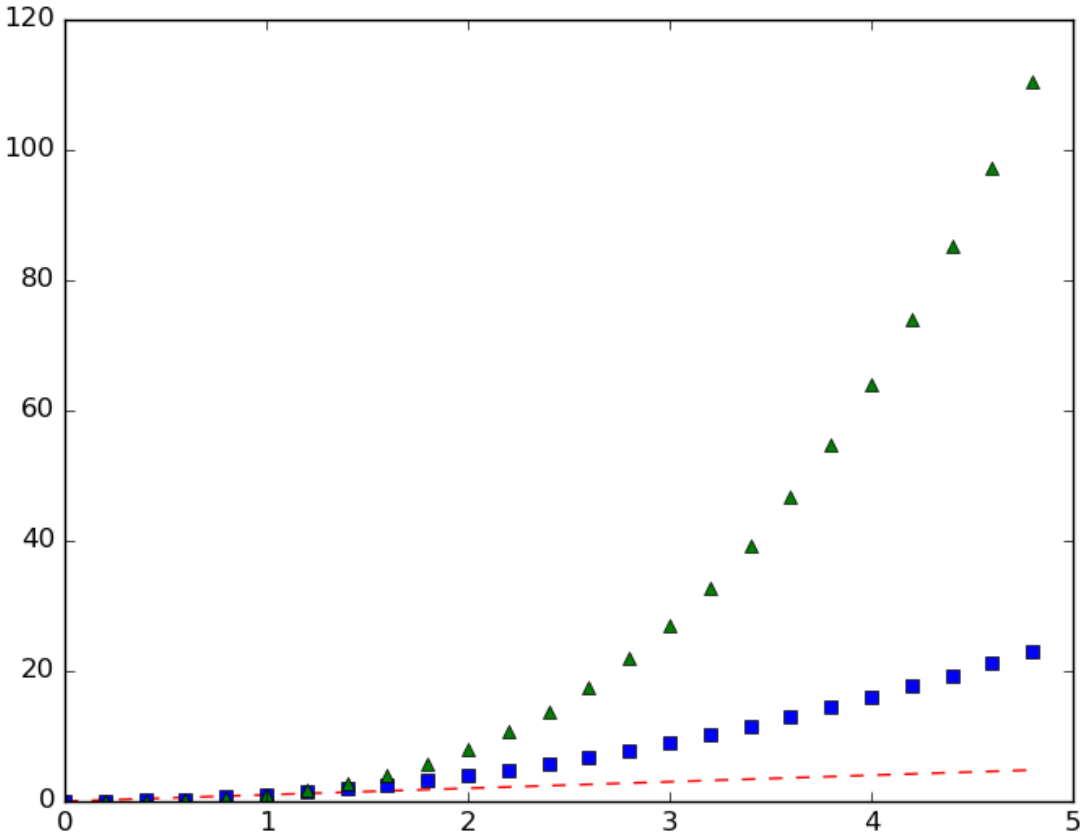
plot() — универсальная команда и в неё можно передавать произвольное количество аргументов. Например, для того, чтобы отобразить y в зависимости от x, можно выполнить команду:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```



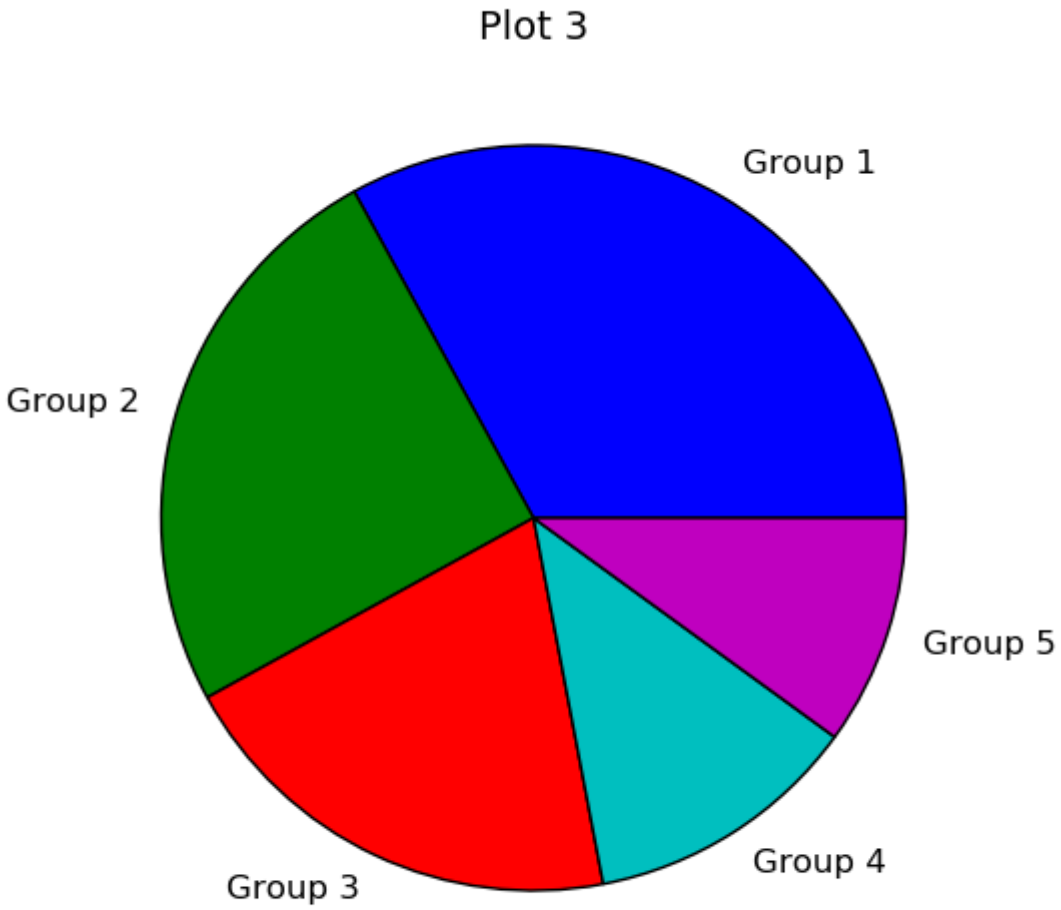
Каждую последовательность можно отобразить своим типом точек:

```
import numpy as np
import matplotlib.pyplot as plt

# равномерно распределённые значения от 0 до 5, с шагом 0.2
t = np.arange(0., 5., 0.2)

# красные чёрточки, синие квадраты и зелёные треугольники
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



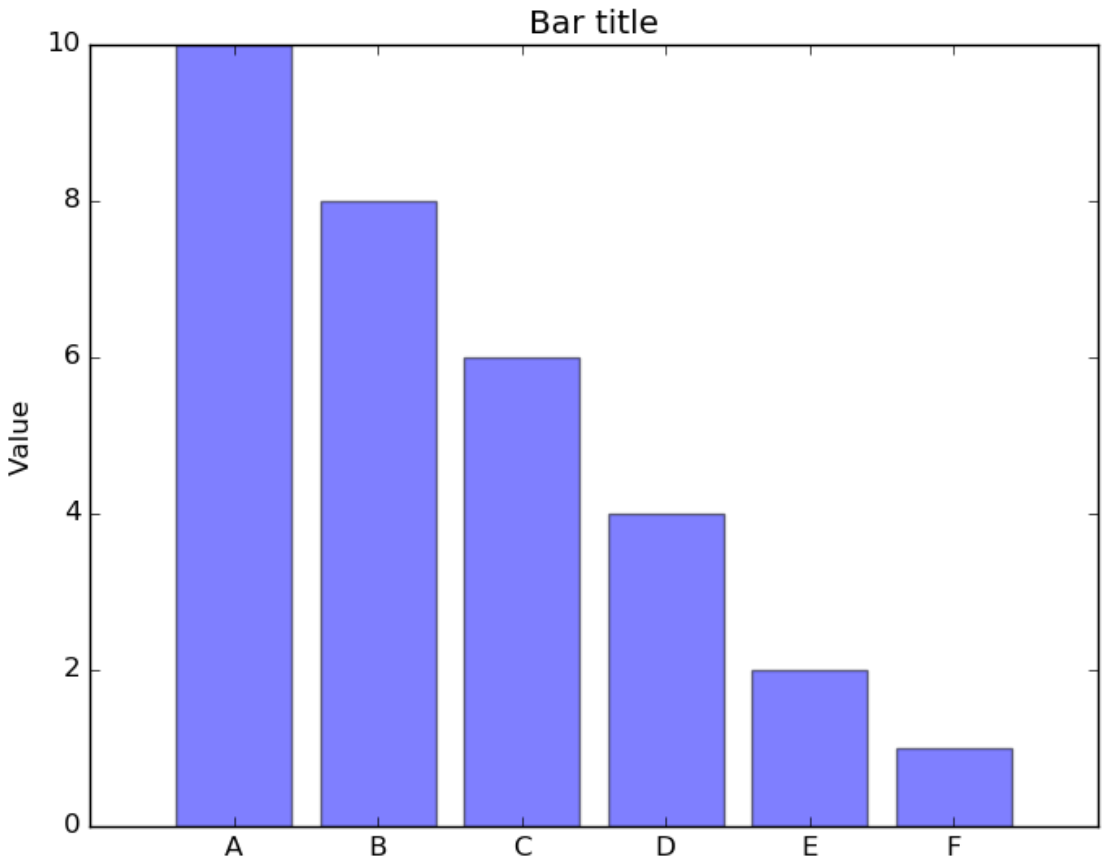
Также в matplotlib существует возможность строить круговые диаграммы:

```
import numpy as np
import matplotlib.pyplot as plt

data = [33, 25, 20, 12, 10]
plt.figure(num=1, figsize=(6, 6))
plt.axes(aspect=1)
plt.title('Plot 3', size=14)
plt.pie(data, labels=('Group 1', 'Group 2', 'Group 3', 'Group 4', 'Group 5'))
plt.show()
```



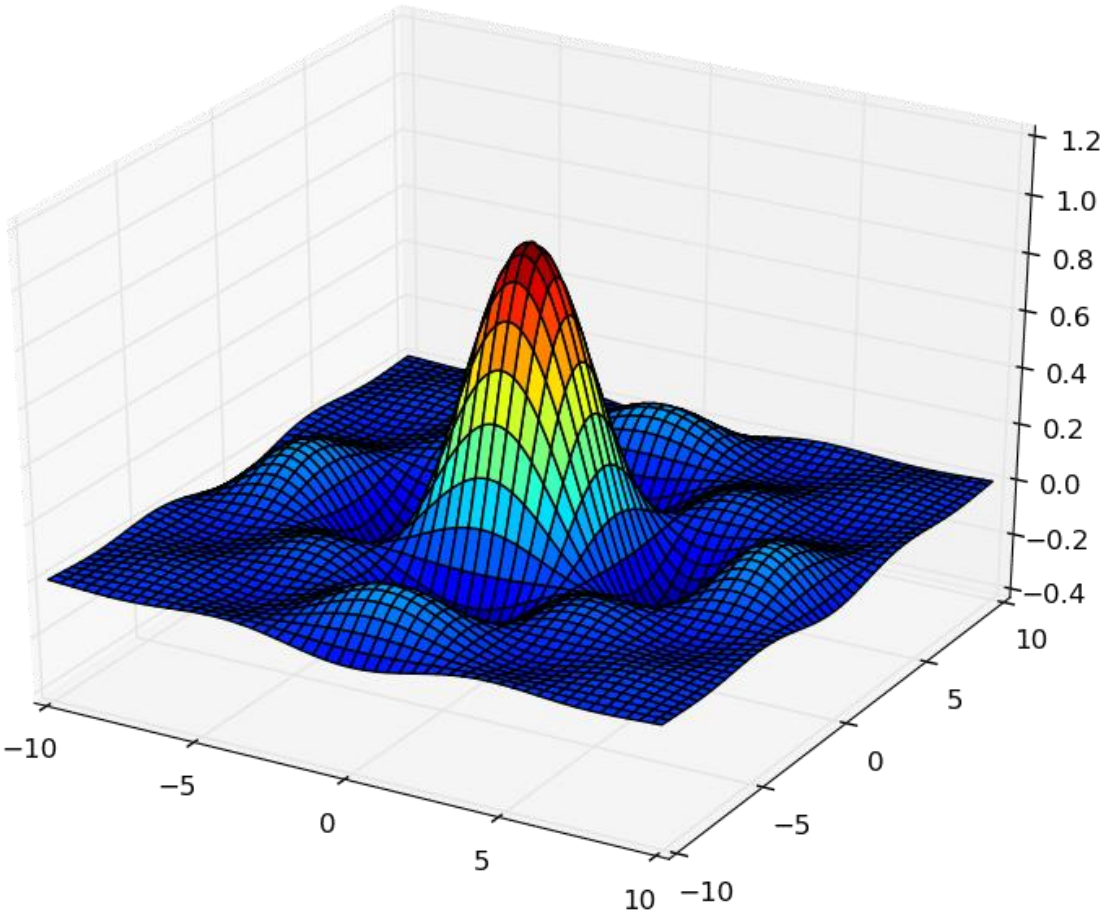
И аналогичным образом столбчатые диаграммы:

```
import numpy as np
import matplotlib.pyplot as plt

objects = ('A', 'B', 'C', 'D', 'E', 'F')
y_pos = np.arange(len(objects))
performance = [10,8,6,4,2,1]

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Value')
plt.title('Bar title')

plt.show()
```



Цветовые карты используются, если нужно указать в какие цвета должны окрашиваться участки трёхмерной поверхности в зависимости от значения Z в этой области. Цветовую карту можно задать самому, а можно воспользоваться готовой. Рассмотрим использование цветовой карты на примере графика функции z(x,y)=sin(x)*sin(y)/(x*y).

```
import pylab
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import numpy

def makeData():
    x = numpy.arange(-10, 10, 0.1)
    y = numpy.arange(-10, 10, 0.1)
    xgrid, ygrid = numpy.meshgrid(x, y)
    zgrid = numpy.sin(xgrid)*numpy.sin(ygrid)/(xgrid*ygrid)
    return xgrid, ygrid, zgrid

x, y, z = makeData()

fig = pylab.figure()
axes = Axes3D(fig)
axes.plot_surface(x, y, z, rstride=4, cstride=4, cmap=cm.jet)
pylab.show()
```



Альтернативой к использованию mpl_toolkits.mplot3d является библиотека plotly, которая позволяет интерактивно взаимодействовать с графиком, поворачивая его или увеличивая некоторую область в пространсте.

Функция eval()

В Python есть встроенная функция eval(), которая выполняет строку с кодом и возвращает результат выполнения:

```
>>> eval("2 + 3*len('hello')")
17
>>>
```

Это очень мощная, но и очень опасная инструкция, особенно если строки, которые вы передаёте в eval, получены не из доверенного источника. Если строкой, которую мы решим скормить eval(), окажется "os.system('rm -rf /')", то интерпретатор честно запустит процесс удаления всех данных с компьютера.

Упражнение №2

Постройте график функции

y(x) = x*x - x - 6

и по графику найдите найдите корни уравнения y(x) = 0. (Не нужно применять численных методов — просто приблизьте график к корням функции настолько, чтобы было удобно их найти.)

Упражнение №3

Постройте график функции

$$\log_{1+\tan\left(\frac{1}{1+\sin^2(x)}\right)}\left(x^2+1\right)\exp\left(-\frac{|x|}{10}\right)$$

Упражнение №4

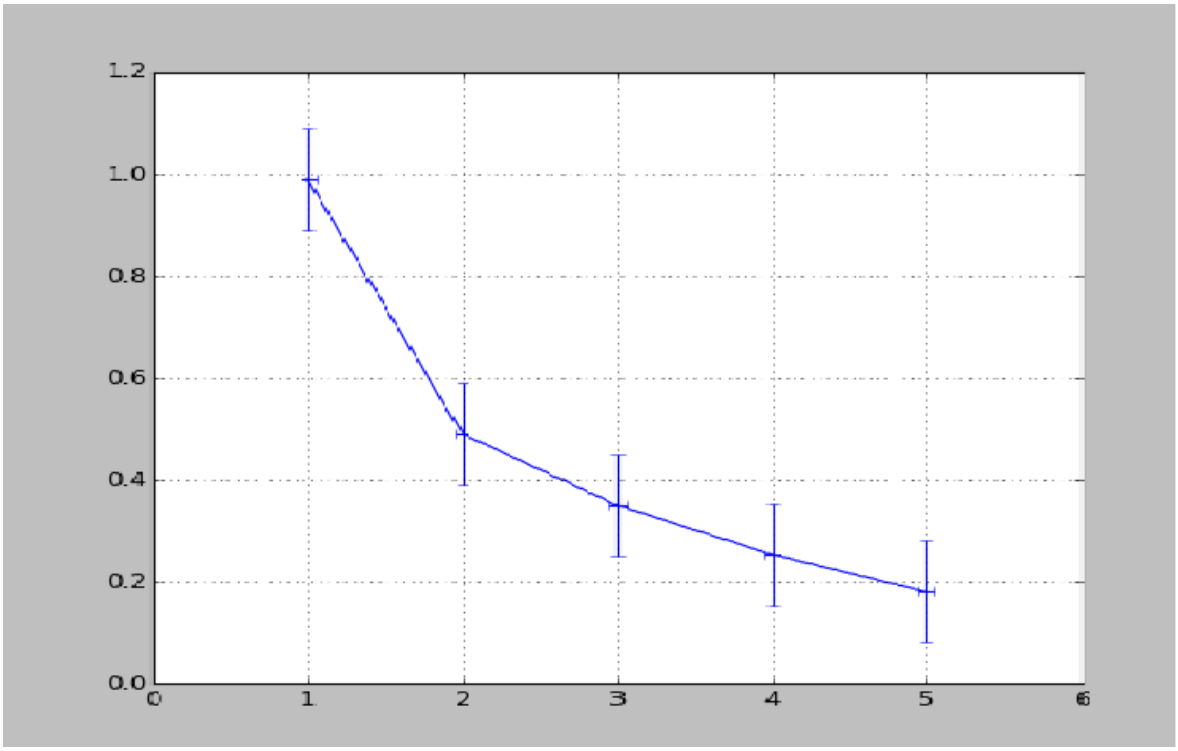
Используя функцию eval() постройте график функции, введённой с клавиатуры. Чтобы считать данные с клавиатуры, используйте функцию input(). Попробуйте включить эффект «рисование от руки» посредством вызова plt.xkcd(). Посольку эта функция применяет некоторый набор настроек, избавиться от которых впоследствие не так просто, удобнее использовать ее как "контекстный менеджер" - это позволяет применить настройки временно, только к определенному блоку кода. Для этого используется ключевое слово with:

```
with plt.xkcd():
    plt.pie([70, 10, 10, 10], labels=('В комментариях', 'В Ираке', 'В Сирии', 'В Афганистане'))
    plt.title('Где ведутся самые ожесточенные бои')
```

Отображение погрешностей

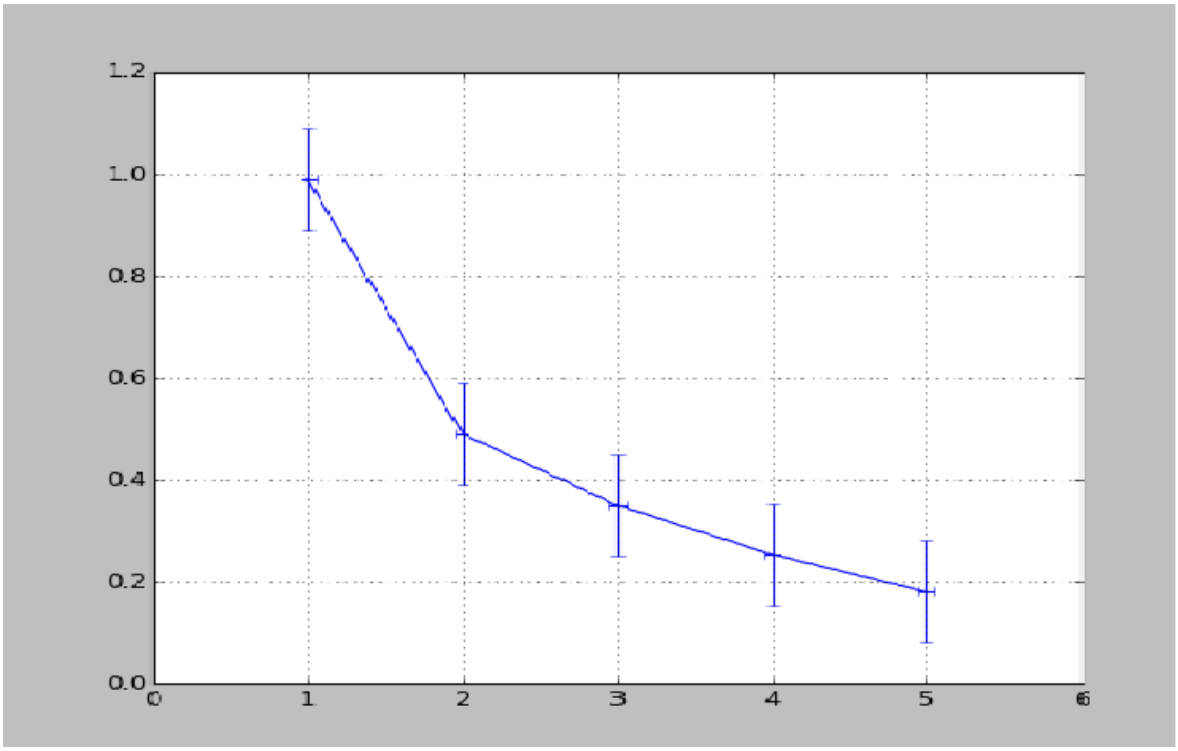
С помощью метода plt.errorbar можно рисовать точки с погрешностями измерений, как для лабораторных работ. Погрешности по осям абсцисс и ординат задаются в параметрах (соответственно) xerr и yerr.

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [0.99, 0.49, 0.35, 0.253, 0.18]
plt.errorbar(x, y, xerr=0.05, yerr=0.1)
plt.grid()
plt.show()
```



Альтернативой для plt.errorbar может слудить plt.fill_between, который заполняет область графика между кривыми, чтобы регулировать прозрачность используется аргумент alpha. Это число из отрезка [0, 1], на которое домножоается интенсивность цвета заполнения между кривыми.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 10, 0.01)
plt.plot(x, x**2, label=r'$f = x^2$')
plt.scatter(x, x**2 + np.random.randn(len(x))*x, s=0.3)
plt.fill_between(x, 1.3*x**2, 0.7*x**2, alpha=0.3)
plt.legend(loc='best')
plt.savefig('figure_fill_between.png')
plt.show()
```



В уже использованном модуле numpy есть метод **polyfit**, позволяющий приближать данные методом наименьших квадратов. Он возвращает погрешности и коэффициенты полученного многочлена.

```
x = [1, 2, 3, 4, 5, 6]
y = [1, 1.42, 1.76, 2, 2.24, 2.5]
p, v = np.polyfit(x, y, deg=1, cov=True)

>>> p
array([0.28517032, 0.80720757])
>>> v
array([[0.00063242, -0.00221348],
[-0.00221348, 0.00959173]])
```

Многочлен задается формулой p(x) = p[0] * x**deg + ... + p[deg]

Для того, чтобы не выписывать каждый раз руками эту формулу для разных степеней, есть функция poly1d, которая возвращает функцию полинома, описанного точками p. Возвращенная функция может принимать на вход не только число, но и список значений, в таком случае, будет вычислено значение функции в каждой точке списка и возвращен список результатов.

```
p_f = np.poly1d(p)
p_f(0.5)
p_f([1, 2, 3])
```

Упражнение №5

Приблизить данные из приведённого примера с погрешностями или свои собственные (из лабораторного практикума по общей физике) многочленами первой и второй степени. Начертить точки с погрешностями и полученные аппроксимационные кривые на одном графике.

Лабораторная работа. Исполнитель "Черепаха" (ч.1).

Запуск интерпретатора Python

В настоящее время существует две версии языка Python: более старая, но пока ещё более распространённая версия 2 и современная версия 3. Мы будем использовать версию 3 данного языка. Именно её необходимо установить дома, скачав данную версию с сайта **www.python.org**.

Запустить интерпретатор python можно из командной строки:

```
python3
```

Будьте внимательны — команда python запустит интерпретатор версии 2, с которым мы работать не будем. В системе Windows можно использовать пункт меню «Python (command line)».

Интерактивный режим

Откройте командную строку и напишите команду python3.

Вы увидите примерно следующее приглашение командной строки:

```
Python 3.7.1 (default, Dec 14 2018, 19:28:38)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Вводите команды и наслаждайтесь результатом. А что можно вводить? Несколько примеров:

```
>>> 2 + 2
4
>>> 2 ** 100
1267650600228229401496703205376
>>> 'Hello' + 'World'
'HelloWorld'
>>> 'ABC' * 10
'ABCABCABCABCABCABCABCABCABCABC'
```

Первая команда вычисляет сумму двух чисел, вторая команда вычисляет 2 в степени 100, третья команда выполняет операцию **конкатенации** для строк, а четвертая команда печатает строку 'ABC', повторенную 10 раз.

Хотите закончить работу с питоном? Введите команду exit() (именно так, со скобочками, так как это — **функция**) или нажмите Ctrl+D.

Программируемый режим

В предыдущей главе мы использовали Python для простых разовых вычислений, используя интерактивный режим. Теперь создадим программу и выполним её целиком.

```
a = 179
b = 197
c = (a ** 2 + b ** 2) ** 0.5
print (c)
```

Здесь мы используем **переменные** — объекты, в которых можно сохранять различные (числовые, строковые и прочие) значения. В первой строке перемен-

ной a присваивается значение 179, затем переменной b присваивается значение 971, затем переменной c присваивается значение арифметического выражения, равного длине гипотенузы. После этого значение переменной c выводится на экран.

Упражнение №1

Откройте произвольный текстовый редактор, например, spyder или Geany. Скопируйте туда текст программы, написанной выше. Сохраните текст в файле с именем hypot.py.

Запустите *терминал*, перейдите в каталог, где лежит файл hypot.py и выполните эту программу:

```
python3 hypot.py
```

Интерпретатор языка Python вместо интерактивного режима выполнит последовательность команд из файла.

При этом значения вычисленных выражений не выводятся на экран (в отличии от интерактивного режима), поэтому для того, чтобы вывести результат работы программы, то есть значение переменной c, нужна функция print().

Базовый синтаксис языка Python 3

Типы данных

Итак, мы видим, что 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 3^{27}.
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 мы использовали *комментарии* – текст, который интерпретатор игнорирует. Комментариями в Pythone является символ # и весь текст после этого символа до конца строки. Желательно писать код так, чтобы комментарии были излишними, однако допускается писать их там, где возникают "призраки" (утверждения или теоремы, которые использованы при написании кода, но не следуют из самого кода). Однако код выше является **плохим примером документации**: комментарии врут, поскольку автором не учтены точки на осях.

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 второ
>=	Больше или равно — условие верно, если первый операнд больше или равен второ
==	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 преобразо-

вании 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 a_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

Команда	Значение
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)
```

Упражнение №2: буква S

Сохраните и выполните предыдущую программу. Убедитесь в том, что черепаха работает.

Упражнение №3: квадрат

Нарисуйте квадрат. Пример:

Упражнение №4: окружность

Нарисуйте окружность. Воспользуйтесь тем фактом, что правильный многоугольник с большим числом сторон будет выглядеть как окружность. Пример:

Упражнение №5: больше квадратов

Нарисуйте 10 вложенных квадратов.

Упражнение №6: паук

Нарисуйте паука с n лапами. Пример n = 12:

Упражнение №7: спираль

Нарисуйте спираль. См. **теорию**. Пример:

Упражнение №8: квадратная «спираль»

Нарисуйте «квадратную» спираль. Пример:

Написание функций

Как было сказано раньше, функции — это своего рода готовые кирпичики, из которых строится программа. До этого момента мы *использовали* стандартные функции (print, input, функции модуля turtle), теперь настало время *написать* функцию:

```
>>> def hello(name):
...     print('Hello, ', name, '!')
...
>>> hello('world')
Hello,  world!
```

Это простейший пример функции, которая принимает в качестве **параметра** имя, а затем выводит на экран сообщение Hello, <имя>. Как видно из примера, функции в языке Python описываются при помощи ключевого слова def:

```
def Имя_функции(параметр_1, параметр_2, ...):
    Блок_операций
```

Так же, как и в случае циклов и условных операторов, **тело** функции выделяется при помощи отступов.

Вызов функции осуществляется по имени с указанием параметров:

```
hello('world')
```

Внутри функции можно использовать те же синтаксические конструкции, что и вне её — циклы, ветвления, можно даже описывать новые функции. Естественно, внутри функции можно работать и с переменными.

Написанная ранее функция имеет особенность — она просто просто выводит текст на экран и не возвращает никакого результата. Многие функции, напротив, занимаются вычислением какого-либо значения, а затем **возвращают** его тому, кто эту функцию **вызвал**. В качестве примера можно рассмотреть функцию для сложения двух чисел:

```
>>> def sum(a, b):
...     return a + b
...
>>> sum(1, 2)
3
>>> sum(5, -7)
-2
```

Для возврата значения из функции используется оператор return: в качестве параметра указывается значение, которое требуется вернуть.

Упражнение №9: правильные многоугольники

Нарисуйте 10 вложенных правильных многоугольников. Используйте функцию, рисующую правильный n-угольник. **Формулы** для нахождения радиуса описанной окружности. Пример:

Упражнение №10: «цветок»

Нарисуйте «цветок» из окружностей. Используйте функцию, рисующую окружность. Пример:

Упражнение №11: «бабочка»

Нарисуйте «бабочку» из окружностей. Используйте функцию, рисующую окружность. Пример:

Упражнение №12: пружина

Нарисуйте пружину. Используйте функцию, рисующую дугу. Пример:

Упражнение №13: смайлик

Нарисуйте смайлик с помощью написанных функций рисования круга и дуги. Пример:

Упражнение №14: звезды

Нарисуйте две звезды: одну с 5 вершинами, другую — с 11. Используйте функцию, рисующую звезду с n вершинами. Пример:

Красивый код на Python

Важная мысль создателя языка Python, Гвидо ван Россума: **код читается намного больше раз, чем пишется**.

Поэтому существуют рекомендации о стиле кодирования PEP8. Они направлены на то, чтобы улучшить читаемость и сделать его согласованным между большим числом проектов. В идеале, весь код будет написан в едином стиле, и любой сможет легко его прочесть.

Генерация случайных чисел

Python порождает случайные числа на основе формулы, так что они не на самом деле случайные, а, как говорят, псевдослучайные.

Модуль **random** позволяет генерировать случайные числа. Прежде чем использовать модуль, необходимо подключить его с помощью инструкции:

```
from random import *
```

Пока вам достаточно знать две функции из этого модуля:

random()	возвращает псевдослучайное число типа float от 0.0 до 1.0
randint(a, b)	возвращает псевдослучайное целое число в промежутке [a, b] включая его границы

Упражнение

Нарисуйте при помощи случайных поворотов и перемещений картину броуновских движений.

Списки

Большинство программ работает не с отдельными переменными, а с набором переменных. Например, программа может обрабатывать информацию об учащихся класса, считывая список учащихся с клавиатуры или из файла, при этом изменение количества учащихся в классе не должно требовать модификации исходного кода программы.

Для хранения таких данных можно использовать структуру данных, называемую в Питоне список. Список представляет собой последовательность элементов, проиндексированных от 0. Список можно задать перечислением элементов в квадратных скобках, например, список можно задать так:

```
Primes = [2, 3, 5, 7, 11, 13]
Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
```

В списке Primes — 6 элементов, а именно, Primes[0] == 2, Primes[1] == 3, Primes[2] == 5, Primes[3] == 7, Primes[4] == 11, Primes[5] == 13. Список Rainbow состоит из 7 элементов, каждый из которых является строкой.
Также как и символы строки, элементы списка можно индексировать отрицательными числами с конца, например, Primes[-1] == 13, Primes[-6] == 2.
Длину списка, то есть количество элементов в нем, можно узнать при помощи функции len, например, len(A) == 6.

Рассмотрим несколько способов создания и считывания списков. Пустой, т.е. не имеющий элементов список, можно создать следующим образом:

```
A = []
```
Для добавления элементов в конец спискаиспользуется функция append. Если программа получает на вход количество элементов в списке n, а потом n элементов списка по одному в отдельной строке, то организовать считывание списка можно так:
```
A = []
for i in range(int(input()):
    A.append(int(input())
```
В этом примере создается пустой список, далее считывается количество элементов в списке, затем по одному считываются элементы списка и добавляются в его конец.

Для списков целиком определены следующие операции: конкатенация списков (добавление одного списка в конец другого) и повторение списков (умножение списка на число). Например:
```
A = [1, 2, 3]
B = [4, 5]
C = A + B
D = B * 3
```
В результате список C будет равен [1, 2, 3, 4, 5], а список D будет равен [4, 5, 4, 5, 4, 5]. Это позволяет по- другому организовать процесс считывания списков: сначала считать размер списка и создать список из нужного числа элементов, затем организовать цикл по переменной i начиная с числа 0 и внутри цикла считывается i-й элемент списка:
```
A = [0] * int(input())
for i in range(len(A)):
    A[i] = int(input())
```
Вывести элементы списка A можно одной инструкцией print(A), при этом будут выведены квадратные скобки вокруг элементов списка и запятые между элементами списка. Такой вывод неудобен, чаще требуется просто вывести все элементы списка в одну строку или по одному элементу в строке. Приведем два примера, также отличающиеся организацией цикла:
```
for i in range(len(A)):
    print(A[i])
```
Здесь в цикле меняется индекс элемента i, затем выводится элемент списка с индексом i.
```
for elem in A:
    print(elem, end = ' ')
```
В этом примере элементы списка выводятся в одну строку, разделенные пробелом, при этом в цикле меняется не индекс элемента списка, а само значение переменной. Например, в цикле for elem in ['red', 'green', 'blue'] переменная elem будет последовательно принимать значения 'red', 'green', 'blue'.

Методы split и join

Выше мы рассмотрели пример считывания списка, когда каждый элемент расположен на отдельной строке. Иногда бывает удобно задать все элементы списка при помощи одной строки. В такой случае используется метод split, определённый в строковом типе:
```
A = input().split()
```

Если при запуске этой программы ввести строку 1 2 3, то список A будет равен ['1', '2', '3']. Обратите внимание, что список будет состоять из строк, а не из чисел. Если хочется получить список именно из чисел, то можно затем элементы списка по одному преобразовать в числа:

```
for i in range(len(A)):
    A[i] = int(A[i])
```

Используя функции языка map и list то же самое можно сделать в одну строку:

```
A = list(map(int, input().split()))
```

Объяснений, как работает этот пример, пока не будет. Если нужно считать список действительных чисел, то нужно заменить тип int на тип float.

У метода split есть необязательный параметр, который определяет, какая строка будет использоваться в качестве разделителя между элементами списка. Например, вызов метода split('.') для строки вернет список, полученный разрезанием этой строки по символам '.'.
Используя «обратные» методы можно вывести список при помощи однострочной команды. Для этого используется метод строки join. У этого метода один параметр: список строк. В результате создаётся строка, полученная соединением элементов списка (которые переданы в качестве параметра) в одну строку, при этом между элементами списка вставляется разделитель, равный той строке, к которой применяется метод. Например, программа

```
A = ['red', 'green', 'blue']
print(' '.join(A))
print(''.join(A))
print('***'.join(A))
```

выведет строки red green blue, redgreenblue и red***green***blue.

Если же список состоит из чисел, то придется использовать еще и функцию map. То есть вывести элементы списка чисел, разделяя их пробелами, можно так:

```
print(' '.join(map(str, A)))
```

Срезы списков

Со списками, так же как и со строками, можно делать срезы. А именно:

A[i:j]	срез из j-i элементов A[i], A[i+1], ..., A[j-1].
A[i:j:-1]	срез из i-j элементов A[i], A[i-1], ..., A[j+1] (то есть меняется порядок элементов).
A[i:j:k]	срез с шагом k: A[i], A[i+k], A[i+2*k],... . Если значение k меньше 0, то элементы идут в противоположном порядке.

Каждое из чисел i или j может отсутствовать, что означает «начало строки»/ или «конец строки»/
Списки, в отличии от строк, являются изменяемыми объектами: можно отдельному элементу списка присвоить новое значение. Но можно менять и целиком срезы. Например:

```
A = [1, 2, 3, 4, 5]
A[2:4] = [7, 8, 9]
```

Получится список, у которого вместо двух элементов среза A[2:4] вставлен новый список уже из трех элементов. Теперь список стал равен [1, 2, 7, 8, 9, 5].

```
A = [1, 2, 4, 5, 6,  7]
A[::-2] = [10, 20, 30, 40]
```

Получится список [40, 2, 30, 4, 20, 6, 10]. Здесь A[::-2] — это список из элементов A[-1], A[-3], A[-5], A[-7], которым присваиваются значения 10, 20, 30, 40 соответственно.

Если **не непрерывному** срезу (то есть срезу с шагом k, отличному от 1), присвоить новое значение, то количество элементов в старом и новом срезе обязательно должно совпадать, в противном случае произойдет ошибка ValueError.

Обратите внимание, A[i] — это **элемент** списка, а не срез!

Операции со списками

операция	действие
x in A	Проверить, содержится ли элемент в списке. Возвращает True или False.
x not in A	То же самое, что not(x in A).
min(A)	Наименьший элемент списка. Элементы списка могут быть числами или строками, для строк сравнение элементов проводится в лексикографическом порядке.
max(A)	Наибольший элемент списка.
sum(A)	Сумма элементов списка, элементы обязательно должны быть числами.
A.index(x)	Индекс первого вхождения элемента x в список, при его отсутствии генерирует исключение ValueError.
A.count(x)	Количество вхождений элемента x в список.
A.append(x)	Добавить в конец списка A элемент x.
A.insert(i, x)	Вставить в список A элемент x на позицию с индексом i. Элементы списка A, которые до вставки имели индексы i и больше сдвигаются вправо.
A.extend(B)	Добавить в конец списка A содержимое списка B.
A.pop()	Удалить из списка последний элемент, возвращается значение удаленного элемента.
A.pop(i)	Удалить из списка элемент с индексом i, возвращается значение удаленного элемента. Все элементы, стоящие правее удаленного, сдвигаются влево.

Генераторы списков

Для создания списка, заполненного одинаковыми элементами, можно использовать оператор повторения списка, например:

```
A = [0] * n
```

Для создания списков, заполненных по более сложным формулам можно использовать list comprehensions или **генераторы списков** (в функциональном программировании они называются "списковые включения"): выражения, позволяющие заполнить новый список значениями некоторого выражения (формулы). Общий вид генератора следующий: [выражение for переменная in список], где переменная — идентификатор некоторой переменной, список — список значений, который принимает данная переменная (как правило, полученный при помощи функции range), выражение — некоторое выражение, которым будут заполнены элементы списка, как правило, зависящее от использованной в генераторе переменной.

Вот несколько примеров использования генераторов.

Квадраты целых чисел:

```
A = [i ** 2 for i in range(1, n + 1)]
```

Вот так можно получить список, заполненный случайными числами от -99 до 99 (используя функцию randint из модуля random):

```
A = [randint(-99, 99) for i in range(n)]
```

Расширенная форма генератора списка позволяет выполнять отсев по значению. Например, здесь список B будет состоять из элементов списка A, которые больше нуля:

```
B = [x for x in A if x > 0]
```

Упражнение

Посмотрите на шрифт для написания почтового индекса на конвертах:

Воспользуйтесь списками кортежей, чтобы задать рисование Черепашкой таких цифр. Нарисуйте на экране индекс 141700.

Работа с текстовыми файлами в Python

До этого для ввода информации мы использовали исключительно клавиатуру. При этом в большинстве случаев данные, считываемые программой, **уже** хранятся на носителе информации в виде **файлов**.

Для каждого файла, с которым необходимо производить операции ввода-вывода, нужно создать специальный объект – поток. Именно с потоками работают программы — использование такого дополнительного слоя **абстракции** позволяет прозрачно работать не только с текстовыми файлами, но и, например, с архивами.

Открытие файла

Открытие файла осуществляется функцией open, которой нужно передать два параметра. Первый параметр — строка, задающая имя открываемого файла. Второй параметр — строка, укахывающая режим октрытия файла.

Существует три режима открытия файлов:

Режим	Описание
"r" (read)	Файл открывается для чтения данных.
"w" (write)	Файл открываетсяна запись, при этом содержимое файла очищается.
"a" (append)	Файл открывается для добавления данных в конец файла.

Если второй параметр не задан, то считается, что файл открывается в режиме чтения.

Функция open возвращает ссылку на **файловый объект**, которую нужно записать в переменную, чтобы потом через данный объект работать с этим файлом. Например:

```
input = open('input.txt', 'r')
output = open('output.txt', 'w')
```

Здесь открыто два файла (один на чтение, другой на запись) и создано два связанных с ними объекта.

Чтение данных из файла

Для файла, открытого на чтение данных, можно несколько методов, позвозволяющих считывать данные. Мы рассмотри три из них: readline, readlines, read.

Метод readline() считывает одну строку из файла (до символа конца строки 'n', возвращается считанная строка вместе с символом 'n'). Если считывание не было успешно (достигнут конец файла), то возвращается пустая строка. Для удаления символа 'n' из конца файла удобно использовать метод строки rstrip(). Например:

```
s = s.rstrip().
```

Метод readlines() считывает все строки из файла и возвращает список из всех считанных строк (одна строка — один элемент списка). При этом символы 'n' остаются в концах строк.

Метод read() считывает все содержимое из файла и возвращает строку, которая может содержать символы 'n'. Если методу read передать целочисленный параметр, то будет считано не более заданного количества байт. Например, считывать файл побайтово можно при помощи метода read(1).

Вывод данных в файл

Данные выводятся в файл при помощи метода write, которому в качестве параметра передается одна строка. Этот метод не выводит символ конца строки 'n' (как это делает функция print при стандартном выводе), поэтому для перехода на новую строку в файле необходимо явно вывести символ 'n'.

Выводить данные в файл можно и при помощи print, если передать функции еще один именованный параметр file. Например:

```
output = open('output.txt', 'w')
print(a, b, c, file=output)
```

Закрытие файла

После окончания работы с файлом необходимо закрыть его при помощи метода close().
Чтобы не забыть это сделать можно воспользоваться *менеджером контекста* with.

```
with open('input.txt') as file:
```

Примеры работы с файлами

Следующая программа считывает все содержимое файла input.txt, записывает его в переменную s, а затем выводит ее в файл output.txt.

```
inp = open('input.txt', 'r')
out = open('output.txt', 'w')
s = inp.read()
out.write(s)
inp.close()
out.close()
```

Для простого считывания содержимого файла можно использовать то, что сам файл является итерируемым по строкам объектом:

```
with open('input.txt') as file:
    for line in file:
        print('line: "', line, '"')
```

Упражнение

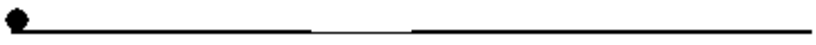
Перенесите описание способа рисования почтовых цифр (списки движений) в файл. Пусть черепаха считывает "шрифт" из файла.

Физическое моделирование материальной точки

Используя оператор turtle.goto(x, y) заставьте черепашку двигаться в равномерном поле тяжести, отталкиваясь от поверхности (уровень y=0).

Основные формулы для расчёта нового местоположения черепшки:

```
x += Vx*dt
y += Vy*dt + ay*dt**2/2
Vy += ay*dt
```



Черепаха как объект

При помощи конструктора turtle.Turtle() можно создать новый объект черепахи. Если поместить эти объекты в список, а потом циклически двигать каждую черепаху на нельшое смещение, возникает эффект одновременного движения:

```
from random import randint
```

```
import turtle


number_of_turtles = 5
steps_of_time_number = 100


pool = [turtle.Turtle(shape='turtle') for i in range(number_of_turtles)]
for unit in pool:
    unit.penup()
    unit.speed(50)
    unit.goto(randint(-200, 200), randint(-200, 200))


for i in range(steps_of_time_number):
    for unit in pool:
        unit.forward(2)
```

При помощи подобного кода заставье черепах вести себя как идеальный газ в сосуде. Если это слишком просто, то как реальный газ.

Список использованных источников

1. МФТИ. Python. Режим доступа: http://cs.mipt.ru/python.