МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Воронежский государственный технический университет»

Кафедра автоматизированного оборудования
машиностроительного производства

КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКАЯ ИНФОРМАТИКА

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ
для студентов направления 15.03.05 «Конструкторско-технологическое обеспечение машиностроительных производств»
(профили «Технология машиностроения», «Металлообратывающие станки и комплексы», «Конструкторско-технологическое обеспечение кузнечно-штамповочного производства»)
всех форм обучения

Воронеж 2021

УДК 621.01. (07)
ББК 34. я 7

Составитель:
ст. преп. С. Л. Новокщенов

Конструкторско-технологическая информатика: методические указания к выполнению лабораторных работ для студентов, обучающихся по направлению 15.03.05 «Конструкторско-технологическое обеспечение машиностроительных производств» (профили «Технология машиностроения», «Металлообрабатывающие станки и комплексы» и «Конструкторско-технологическое обеспечение кузнечно-штамповочного производства») всех форм обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост.: С. Л. Новокщенов. – Воронеж: Изд-во ВГТУ, 2021. – 30 с.

Методические указания содержат сведения, необходимые для выполнения лабораторных работ, а также приведены основные положения курса, и список литературы.

Предназначены для студентов, обучающихся по направлению 15.03.05 «Конструкторско-технологическое обеспечение машиностроительных производств» (профили «Технология машиностроения», «Металлообрабатывающие станки и комплексы» и «Конструкторско-технологическое обеспечение кузнечно-штамповочного производства») всех форм обучения.

Методические указания подготовлены в электронном виде и содержатся в файле МУ ЛР КТИ.pdf.

Ил. 6; Табл. 3; Библиогр. 2 назв.

УДК 621.01. (07)
ББК 34. я 7

Рецензент: *А. В. Демидов, канд. техн. наук, доцент кафедры автоматизированного оборудования машиностроительного производства ВГТУ*

Издается по решению редакционно-издательского совета Воронежского Государственного Технического Университета

ВВЕДЕНИЕ

Небывалые скорости развития науки, техники и технологии породили стремительное развитие вычислительной техники, произошедшее за последнее время, объясняется тем, что сразу во многих областях инженерной деятельности возникла потребность в резком ускорении информационно-вычислительных процессов. ЭВМ при этом позволяет принимать решения на основе огромного количества данных в кратчайший срок и с большой точностью, так как иногда бессмысленно производить точные вычисления, потому как время, затраченное на них, больше отводимого текущей ситуацией.

Для решения инженерных задач, связанных с проектированием или управлением производственных процессов, в настоящее время имеется множество средств (как классических языков программирования, так и средств визуального программирования и представления информации).

Предлагаемый лабораторный практикум позволит студентам ознакомиться с возможностями современного языка программирования Python, который предназначен для представления полученной информации в наглядной понятной форме.

ОРГАНИЗАЦИЯ ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Занятия в лаборатории проводятся под руководством преподавателя. Для проведения лабораторных занятий группа делится на подгруппы (по 10 - 12 человек), постоянный состав которых сохраняется до окончания всего лабораторного практикума. Лабораторные работы выполняется студентами самостоятельно. По результатам выполненных работ оформляется отчет. По окончании лабораторного практикума каждый студент должен сдать зачёт. При сдаче зачёта студент обязан:

1. Знать целевое назначение работы и уметь объяснить порядок и технику её выполнения.
2. Знать устройство, приемы управления и настройку оборудования, приборов и программных средств, применяемых в работе.
3. Понимать физический и практический смысл полученных результатов.
4. Предъявить отчёт с записями со всеми необходимыми расчётами, эскизами, графиками и выводами по каждой выполненной работе.

ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Перед началом лабораторных занятий студенты знакомятся с содержанием лабораторного практикума, организацией и режимом занятий, правилами техники безопасности. Распределение обязанностей внутри подгруппы производится студентами с соблюдением принципа равного участия в работе каждого студента.

Студенты должны:

1. Изучить самостоятельно методику выполнения работы и ознакомиться с организацией рабочего места.
2. Ознакомиться под руководством преподавателя или лаборанта с устройством лабораторного оборудования и его управлением.
3. Категорически запрещается самостоятельный пуск оборудования и пользование без ведома преподавателя или лаборанта.
4. Изучить правила техники безопасности.
5. Произвести под руководством преподавателя или лаборанта настройку оборудования и приборов.
6. Выполнить самостоятельно необходимые учебные задания в соответствии с методикой. Результаты занести в рабочую тетрадь.
7. После окончания работы рабочее место сдать лаборанту.
8. Провести анализ полученных результатов и сделать выводы по работе. Оформить и сдать преподавателю отчет.

ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет по работе оформляется на бумаге стандартного формата (формат А4). Отчет брошюруется в общую тетрадь. Отчет представляется в печатном виде. Коллективное составление и сдача отчетов не допускается.

Отчет по лабораторной работе должен быть выполнен в текстовом редакторе Microsoft Word 2010 или выше и содержать: титульный лист, название темы работы, цели работы, перечень технических и программных средств, необходимых для выполнения лабораторной работы; краткое описание исследуемого вопроса; алгоритм программы; исходные данные варианта; распечатку полученных в ходе расчета значений; выводы, содержащие анализ проведенной работы.

В выводах дается краткое объяснение сущности полученных результатов. Выводы должны быть краткими и отвечать на вопросы, поставленные в лабораторной работе.

ТЕХНИКА БЕЗОПАСНОСТИ ПРИ РАБОТЕ СТУДЕНТОВ В ЛАБОРАТОРИИ

Для того чтобы уберечь себя и товарищей от несчастного случая, а государственное имущество от аварии, необходимо хорошо знать и полностью выполнять правила внутреннего распорядка, техники безопасности и пожарной безопасности.

К лабораторным работам допускаются студенты, которые ознакомились с общими конкретными требованиями техники безопасности и прошли соответствующий инструктаж. Проведение инструктажа и проверка знаний правил техники безопасности должны быть зарегистрированы соответствующими записями в лабораторном журнале. Конкретные требования техники безопасности при проведении той или иной работы изложены в описании к лабораторным работам.

Лабораторная работа №1

Интегрированная среда разработки (4 часа)

Цель работы: изучить назначение, ознакомиться с основными возможностями и принципами работы в интегрированной среде программирования.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Интегрированная среда разработки (англ. *Integrated Development Environment*) представляет собой систему программных средств, используемую программистами для разработки программного обеспечения. В настоящее время наиболее широко используемой IDE является Microsoft Visual Studio (рис. 1).

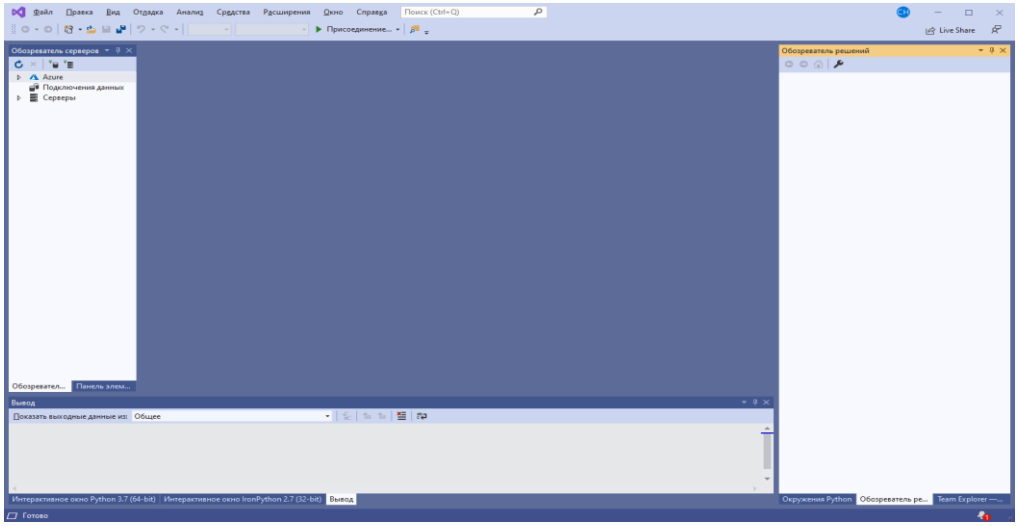


Рис. 1. Интерфейс IDE Microsoft Visual Studio

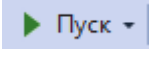
После запуска Visual Studio можно начать новый проект, выполнив команду **Файл – Создать – Проект** при этом программисту доступно несколько языковых средств для разработки приложения. Создавая или открывая приложение или просто отдельный файл, система Visual Studio использует концепцию ***решения (solution)*** для связывания всех компонентов в единое целое. Для начинала освоения языка разберем пример создания наиболее простого типа программы – приложения, которое работает в командной строке – текстовом интерфейсе операционной системы Microsoft Windows.

После выбора типа проекта вид окна Visual Studio изменится - в нем появится окно текстового редактора, в котором осуществляется ввод и редактирование программы. Теперь, чтобы написать программу, возникает необходимость в знании ее основных элементов применительно к используемому языку программирования. Создадим первую программу, которая выводит в командную строку некий текст. Для этого достаточного ввести следующую строчку (рис. 2).

```
print('Hello Python!')
```

Рис. 2. Код первой программы

Запуск созданного программы

Теперь нашу первую программу можно попробовать запустить на выполнение, для чего достаточно нажать кнопку 

Результат работы программы

По выполнению этого кода в командной строке должно появиться (рис. 3).

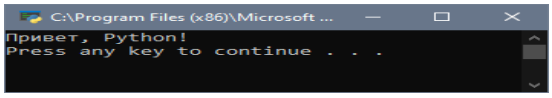


Рис. 3. Сообщение командной строки после изменения кода программы

Кроме операторов ввода и вывода язык Python поддерживает набор самых обычных математических операций:

Таблица 1

Поддерживаемые математические операции

№	Операция	Описание
1	a + b	Сложение
2	a – b	Вычитание
3	a * b	Умножение
4	a / b	Деление
5	a // b	Получение целой части от деления
6	a % b	Остаток от деления
7	-a	Смена знака числа
8	abs(a)	Модуль числа
9	divmod(a, b)	Пара (a // b, a % b)
10	a**b	Возведение в степень
11	pow(a, b, [c])	ab по модулю (если модуль задан)

Задание

1) Разработайте программу на языке Python. Варианты для выполнения задания указаны в табл. 2.

Таблица 2

№	Текст	№	Текст
1	Режимы резания на Python	14	Режимы резания на Python
2	Станок токарный, режимы на Python	15	Станок фрезерный, режимы на Python
3	Станок шлифовальный, режимы на Python	16	Станок фрезерный ЧПУ, режимы на Python
4	Станок фрезерный, режимы на Python	17	Станок сверлильный, режимы на Python
5	Станок сверлильный, режимы на Python	18	Режимы резания на Python
6	Станок токарный ЧПУ, режимы на Python	19	Станок токарный ЧПУ, режимы на Python
7	Режимы резания на Python	20	Станок сверлильный, режимы на Python
8	Станок фрезерный ЧПУ, режимы на Python	21	Станок фрезерный, режимы на Python
9	Станок токарный, режимы на Python	22	Станок фрезерный ЧПУ, режимы на Python

Окончание таблицы 2

№	Текст	№	Текст
10	Станок сверлильный, режимы на Python	23	Станок шлифовальный, режимы на Python
11	Станок фрезерный, режимы на Python	24	Станок токарный ЧПУ, режимы на Python
12	Станок токарный ЧПУ, режимы на Python	25	Режимы резания на Python
13	Станок сверлильный, режимы на Python		

2) Разработайте программу на языке Python, выполнявшую одно или несколько арифметических действий.

Лабораторная работа № 2

Автоматизация конструкторско-технологических расчетов (4 часа)

Цель работы: изучить структуру программы на языке Python, освоить работу с составными операторами.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Последовательность действия в программах Python описываются последовательными строками. Сложные алгоритмы реализуются в языке Python при помощи структур ветвления и цикла, которые поддерживаются операторами:

1. **if** - условное ветвление;
2. **while** - цикл с условием;
3. **for** - совместные циклы (циклы по коллекциям).

Программа на языке Python состоит из определенного набора инструкций, выполняемых последовательно. При появлении уровней вложенности используются отступы. Каждый уровень вложенности смещается на четыре пробельных отступа (распространенное соглашение). Инструкция обычно завершается в конце строки, в которой она находится. Перенос

возможен либо при помощи символа \ (метод устарел, не рекомендуется использовать), либо в том случае, когда для интерпретатора четко указано начало и конец инструкции, например при помощи открытой скобки из синтаксической пары.

```
L = ['one',
  'two',
 'three']
```

Базовые процессы в ЭВМ могут быть следующих основных типов (рис. 4).

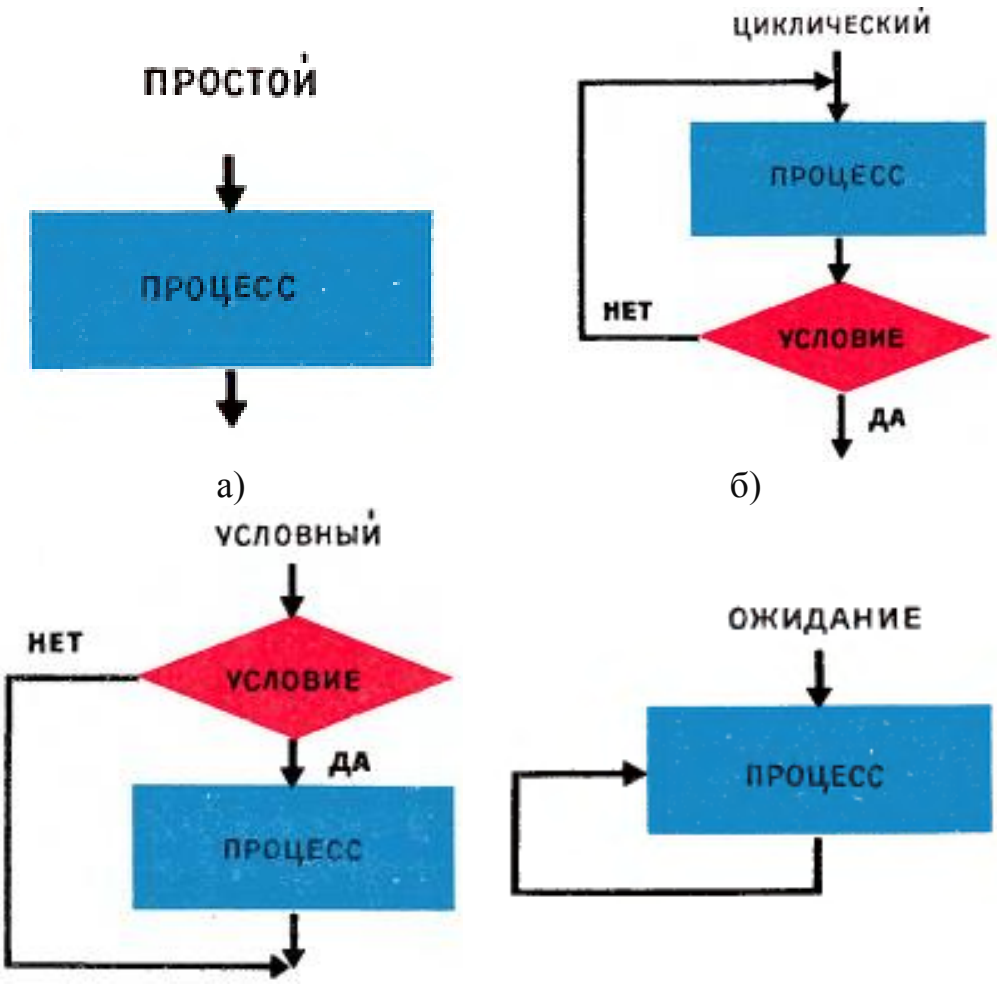


Рис. 4. Базовые процессы, происходящие в ЭВМ

Последовательность действия в программах Python описываются последовательными строками. Сложные алгоритмы реализуются в языке Python при помощи структур ветвления и цикла, которые поддерживаются операторами:

4. **if** - условное ветвление;
5. **while** - цикл с условием;
6. **for** - совместные циклы (циклы по коллекциям).

Задание

1) Разработайте программу на языке Python. Варианты для выполнения задания указаны в табл.

Таблица 3

№	Задача	
1	$n = \frac{1000 \cdot V}{\pi \cdot D}$	Определить число оборотов шпинделя;
2	$t = \frac{60 \cdot F \cdot k}{N}$	Выбрать тип производства;
3	$t = (t_0 + t_в) \cdot k_{отв}$	Найти время обработки каждого отверстия детали;
4	$n = \frac{t_c}{t_p} + 1$	Число станков, обслуживаемых одним рабочим;
5	$\eta_З = {k_p}/{k_п}$	Определить коэффициент загрузки станка, %;
6	$P = f \cdot p \cdot \pi \cdot d \cdot l$	Сила запрессовки при натяге, Н;
7	$S_a = \frac{S_{маш} \cdot a}{F}$	Расходы по амортизации оборудования (в руб.);
8	$A = {T_{авт}}/{T_{тх}}$	Показатель уровня автоматизации технологического процесса, %;
9	$V = {A}/{T^m}$	Вычислить скорость резания;
10	$T = \frac{1-m}{m} \cdot t_и$	Определить стойкость режущего инструмента, час.

Лабораторная работа № 3

Алгоритмы на Python. Условный оператор (4 часа)

Цель работы: научить принципы создания и применения условного оператора в программе.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Условные операторы

В Python реализован следующий синтаксис при использовании условных операторов при помощи оператора if:

```
if <Логическое_выражение_1>:
```

```
    <Набор_инструкций_1>
```

В том случае, когда Логическое_выражение_1 возвращает True, то выполняется Набор_инструкций_1, расположенный после символа двоеточия и смещенный на один уровень. Если должен выполняться определенный набор инструкции в том случае, когда Логическое_выражение_1 возвращает False, то используется ключевое слово else:

```
if <Логическое_выражение_1>:
    <Набор_инструкций_1>
        else:
    <Набор_инструкций_2>
```

Для того, чтобы проверить несколько логических выражений в условном операторе может использоваться ключевое слово elif. Блоков elif может быть несколько:

```
if <Логическое_выражение_1>:
    <Набор_инструкций_1>
elif <Логическое_выражение_2>:
    <Набор_инструкций_2>
elif <Логическое_выражение_3>:
        pass
        else:
    <Набор_инструкций_4>
```

Вначале проверяется Логическое_выражение_1. Если оно возвращает False, то последовательно проверяются логические выражения в блоках elif. Если все логические выражения возвращают False, то выполняется Набор_инструкций_4, следующий за ключевым словом else.Ключевое слово pass используется в том случае, если действия не должны выполняться. Добавим в нашу первую программу, описанную в предыдущей лабораторной работе, условие проверки на не нулевое значение введенного числа, которое попадет в знаменатель при нахождении частного, тем самым мы учтем, что на ноль делить нельзя. Условный оператор, необходимый для этой проверки примет следующий вид (рис.5):

```
if b!=0: rezult2=a/b
else: print('Введите число, отличное от нуля...')
b=float(input('Введите второе число '))
```

а)

```
if b==0: b=float(input('Введите число, отличное от нуля, повторите ввод второго числа: '))
if b!=0:
    rezult=a+b
    rezult1=a-b
    rezult2=a/b
    rezult3=a*b
else:
    rezult=a+b
    rezult1=a-b
    rezult2=a/b
    rezult3=a*b
```

б)

Рис. 5. Условный оператор

Задание

1) Для программы, разработанной в лабораторной работе №2 примените условный оператор (проверку возможности получения результата вычисления по значению переменной).

Лабораторная работа № 4

Алгоритмы на Python. Операторы цикла (4 часа)

Цель работы: научить принципы создания и применения операторов цикла в программе.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Циклы и итерации

Цикл **while** служит для ***повторения определенного набора инструкций*** при соблюдении определенного условия и имеет следующий синтаксис:

```
while <Логическое_выражение_1>:
    <Набор_инструкций_1>
```

После ключевого слова while располагается Логическое_выражение_1. Набор_инструкций_1 смещен на одни уровень и будет выполняться до тех пор, пока Логическое_выражение_1 будет возвращать True. Соответственно цикл while True: будет выполняться бесконечно. Для управления выполнением цикла могут использоваться ключевые слова break и continue. Ключевое слово break приведет к выходу из цикла, а continue к следующей итерации цикла.

```
i=0
while True:
    i+=1
    if i==1:
        continue
    elif i==4:
        break
    print(i)
```

В данном примере при i==1 будет произведен переход к следующей итерации и значение 1 не будет выведено на экран, а при i==4 будет выполнен выход из цикла.

Расширить функционал цикла while можно за счет использования блока else:

```
while <Логическое_выражение_1>:
    <Набор_инструкций_1>
else:
    <Набор_инструкций_2>
```

Набор_инструкций_2 будет выполняться только в случае нормального завершении цикла – без выполнения инструкции break.

Ниже приведен пример программы, вычисляющей факториал числа при помощи цикла while. Переменная `number` содержит целое число, для которого необходимо вычислить факториал, `i` – счетчик, `factorial` – хранит результат вычислений. Цикл вычисления факториала будет выполняться до тех пор, пока значение счетчика меньше или равно значению числа, для которого вычисляется факториал.

```
number = int(input("Введите число: "))
i = 1
factorial = 1
while i <= number:
    factorial *= i
    i += 1
print("Факториал числа", number, "равен", factorial)
```

Цикл for используется для работы с любыми итерируемыми объектами, например списками, кортежами, строками и т. д.

```
for <Переменная> in <Итерируемый_объект>:
    <Набор_инструкций>
```

Набор_инструкций смещен на одни уровень и будет выполняться до тех пор, пока Переменная будет последовательно принимать значения из Итерируемый_объект. Также как и в цикле while может дополнительно использоваться блок else.

```
for i in 'Hello':
    print(i)
else:
    print('Цикл успешно завершен')
```

В данном примере переменная i будет последовательно принимать значения символов в строке ‘Hello’ и выводить их на экран. После завершения работы цикла будет выведено сообщение 'Цикл успешно завершен' на экран.

Если какое-то действие необходимо выполнить n-ное количество раз, то может использоваться функция range, которая возвращает ряд чисел в зависимости от переданных аргументов.

- range(stop): возвращает все целые числа от 0 до stop
- range(start, stop): возвращает все целые числа в промежутке от start (включая) до stop (не включая).
- range(start, stop, step): возвращает целые числа в промежутке от start (включая) до stop (не включая), которые увеличиваются на значение step

Примеры вызовов функции range:

```
range(5) # 0, 1, 2, 3, 4
range(1, 5) # 1, 2, 3, 4
range(2, 10, 2) # 2, 4, 6, 8
range(5, 0, -1) # 5, 4, 3, 2, 1
```

Пример реализации функции range:

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

Если элементы, извлекаемые в процессе итерации представляют собой последовательности, то их можно распаковать в переменные.

```
list_1 = [(0, 0),(1, 1),(2, 2)]
for i, j in list_1:
    print(i, j)
```

Здесь кортежи списка list_1 распаковываются в переменные i, j. Если при выполнении итераций необходимо иметь доступ к порядковому номеру, то в таком случае может использоваться функция enumerate.

```
list_2 = ['Токарный', 'Сверлильный', 'Фрезерный']
for n, i in enumerate(list_2):
    print(n, i)
```

Вызов функции enumerate(list_2) возвращает итератор, содержащий последовательность кортежей (0, list_2[0]), (1, list_2[1]), (2, list_2[2]), который распаковываются в переменные n, i.При работе одновременно с несколькими последовательностями удобно использовать функцию zip, которая объединяет соответствующие элементы последовательностей в кортежи, которые в дальнейшем можно распаковать. При несоответствии размеров входных последовательностей, выходная будет иметь длину меньшей их них.

```
list_2 = ['Токарный', 'Сверлильный', 'Фрезерный']
list_3 = ['005', '010','015', '020']
for n, i in zip(list_3, list_2):
    print(n, i)
```

Ниже приведен пример программы, вычисляющей факториал числа при помощи цикла for

```
number = int(input("Введите число: "))
factorial = 1
for i in range(1, number+1):
    factorial *= i
print("Факториал числа", number, "равен", factorial)
```

Задание

1) Для программы, разработанной в лабораторной работе №2 примените одну из приведенных конструкций циклов для получения ряда значений.

Лабораторная работа № 5

Алгоритмы на Python. Функции
(4 часа)

Цель работы: научить принципы создания и применения функций в программе.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Функции

Функции представляют собой наиболее базовую программную структуру Python, которая предназначена для доведения до максимума многократного использования кода. Функции — это почти универсальная методика структурирования программ. Функции исполняют две основные роли во время разработки.

1) Доведение до максимума многократного использования кода и сведение к минимуму избыточности. Как и в большинстве языков программирования, функции Python являются простейшим способом упаковки логики, которую вы, возможно, захотите применять более чем в одном месте и более одного раза.

2) Процедурная декомпозиция. Функции также предлагают инструмент для разбиения систем на части с хорошо определенными ролями. Оператор def создает объект функции и присваивает его имени. Общий синтаксис функции выглядит следующим образом:

```
def <Имя_функции>(<Аргументы>):
        <Набор_инструкций>
    return <Возвращаемое_значение>
```

После ключевого слова def указывается имя функции, которому присваивается объект функции, а также кортеж из нуля или более аргументов, передаваемых при вызове функции. Набор_инструкций является телом функции, т. е. кодом, который Python выполняет каждый раз, когда функция позже вызывается. Оператор return в Python может появляться, где угодно в теле функции; по достижении он заканчивает вызов функции и посылает результат обратно вызывающему коду. Оператор return состоит из необязательного выражения с объектным значением, которое дает результат функции. Если значение опущено, тогда return отправляет обратно None. Функции могут также содержать операторы yield, которые предназначены для производства серии значений с течением времени. Оператор def создает функцию, но не

вызывает ее. После выполнения def функцию можно вызывать (выполнять) в своей программе, добавляя к имени функции круглые скобки:

```
def sm(x, y):
 return x + y
print(sm(2,4))
```

Порядок следования аргументов в вызове функции должен совпадать с порядком следования аргументов в определении функции. Аргументы функции могут иметь значения по умолчанию.

```
def func(a, b, c = 2): # c - необязательный аргумент
    return a + b + c
```

Если в объявлении функции присутствует аргумент со значением по умолчанию, этот аргумент и все следующие за ним считаются необязательными.

```
print(func(1,3))
```

Значения аргументов по умолчанию всегда связываются с теми самыми объектами, которые использовались в качестве значений в объявлении функции. Например:

```
a=10
def foo(x = a):
    return x
a=5
print(foo())
```

Если перед последним аргументом в определении функции добавить символ звездочки *, функция сможет принимать переменное число аргументов:

```
def summator(*args):
    sum = 0
    for i in args:
        sum += i
    return summ
print(summator(1,3,5,6,78,3,234,52,52,2))
```

В данном случае все дополнительные аргументы будут помещены в переменную args в виде кортежа. Кроме того, имеется возможность передавать функциям аргументы, явно указывая их имена и значения. Такие аргументы называются именованными аргументами. Например:

```
def func(w, x, y, z):
    tmp=(x + y)**z[1]
    return w + str(tmp)
#Вызов с именованными аргументами
print(func(x = 3, y = 22, w = 'hello ', z = [1,2]))
#hello 625
```

При вызове с именованными аргументами порядок следования аргументов не имеет значения. Однако при этом должны быть явно указаны имена всех обязательных аргументов, если только они не имеют значений по умолчанию.

В одном вызове функции допускается одновременно использовать позиционные и именованные аргументы, при соблюдении следующих условий: первыми должны быть указаны позиционные аргументы, должны

быть определены значения для всех обязательных аргументов и ни для одного из аргументов не должно быть передано более одного значения.

Например:

```
foo(‘hello’, 3, z = [1,2], y = 22)
```

Если последний аргумент в объявлении функции начинается с символов **, все дополнительные именованные аргументы (имена которых отсутствуют в объявлении функции) будут помещены в словарь и переданы функции. Это обеспечивает удобную возможность писать функции, способные принимать значительное количество параметров, описание которых в объявлении функции выглядело бы слишком громоздко. Например:

```
def make_table(data, **parms):
#Получить параметры из аргумента parms (словарь)
fgcolor = parms.pop('fgcolor')
bgcolor = parms.pop('bgcolor')
  width = parms.pop('width')
make_table(items, fgcolor='black', bgcolor='white', width=400)
```

Задание

1) Для программы, разработанной в лабораторной работе №2 модифицируйте программу с помощью определения функции.

Лабораторная работа № 6

Работа с файлами и файловой системой (4 часа)

Цель работы: научить методику работы с файловой системой, используя модуль os, изучить списки, строки, методы форматирования строк, используя оператор %.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Импорт модулей

Модули в Python представляют собой единицы организации программ наивысшего уровня, которые упаковывают программный код и данные для многократного использования и предоставляют изолированные пространства имен, сводящие к минимуму конфликты имен переменных внутри программ. Каждый файл является модулем, а модули импортируют другие

модули, чтобы задействовать определяемые в них имена. Модули могут также соответствовать расширениям, написанным на внешнем языке, таком как C, Java или C#, и даже каталогам в импортированных пакетах.

import позволяет клиенту- (импортеру) извлечь модуль как единое целое.

```
import random
print(random.random())
```

Для гибкости работы с пространством имен может использоваться as:

```
import random as rnd
print(rnd.random())
```

from позволяет клиентам извлекать отдельные имена из модуля.

```
from random import random
print(random())
```

* позволяет клиентам извлекать все имена из модуля.

```
from random import *
print(randint(0,10))
```

Модули предоставляют простой способ организации компонентов в систему, выступая в качестве изолированных пакетов переменных, которые известны как пространства имен. Все имена, определенные на верхнем уровне файла модуля, становятся атрибутами объекта импортированного модуля. Как было показано ранее, импортирование обеспечивает доступ к именам в глобальной области видимости модуля. Таким образом, глобальная область видимости файла модуля превращается в пространство имен объекта модуля, когда файл модуля импортируется. В итоге модули Python позволяют связывать индивидуальные файлы в более крупную программную систему.

Модуль os

Модуль os содержит функции для работы с различными операционными системами, причем их поведение, как правило, не зависит от текущей ОС.

Для импорта модуля используется следующая команда:

```
import os
```

Ниже представлены основные функции модуля os для работы с файловой системой и примеры их использования.

- `os.name # Возвращает имя операционной системы.` Обращение к элементам файловой системы происходит относительно рабочего каталога (того каталога, где расположена запускаемая программа).
- `os.getcwd() # Возвращает текущий рабочий каталог.` Если необходимо работать с файловой системой в другом каталоге, то необходимо указать путь к нему:
  1) Абсолютный путь (с учетом корневого каталога);
  2) Относительный путь (путь с учетом текущего каталога)
- `os.chdir(path) # Меняет текущий рабочий каталог.`
- `os.chdir('..') # Изменит рабочий каталог на один уровень выше.`
- `os.listdir(path )# Возвращает список файлов и директорий в папке.`

Переменная dir_fd может быть не реализована в используемой операционной системе, что вызовет исключение NotImplementedError

```
os.mkdir(path, mode=511, *, dir_fd=None) # Создаёт директорию.
```

Если создаваемая директория существует, то будет вызвано исключение FileExistsError

```
os.remove(path, *, dir_fd=None) # Удаляет файл.
```

При отсутствии файла вызовет исключение PermissionError

```
os.rename(src, dst, *, src_dir_fd=None, dst_dir_fd=None) # Переимено
вывает файл или директорию.
```

При отсутствии файла или каталога вызовет исключение FileNotFoundError

```
os.replace(src, dst, *, src_dir_fd=None, dst_dir_fd=None) # Переимен
овывает файл или каталог с принудительной заменой.
```

При отсутствии файла или каталога вызовет исключение FileNotFoundError

```
os.rmdir(path, *, dir_fd=None) # Удаляет директорию.
```

При отсутствии каталога вызовет исключение FileNotFoundError

```
os.walk(top, topdown=True, onerror=None, followlinks=False) # Генера
тор дерева каталогов.
```

Для каждого каталога в корневом древе каталога возвращает кортеж, состоящий из трех элементов - dirpath, dirnames, filenames, где:

–dirpath – это строка, путь к каталогу;

–dirnames – это список подкаталогов в dirpath;

–filenames – это список имен файлов, не входящих в каталог dirpath.

Обратите внимание, что имена файлов в filenames не содержат компонентов пути. Чтобы получить полный путь к файлу или каталогу в dirpath:

```
os.path.join(dirpath, name)
```

Данный пример выведет абсолютный путь ко всем файлам, располагающихся в 'Documents' и подкаталогах:

```
path='Documents'
    for root, dirs, files in os.walk(path):
        for file in files:
            print(os.path.join(root,file))
```

Модуль os.path

os.path является вложенным модулем в модуль os, и реализует некоторые полезные функции для работы с путями.

```
os.path.join(path, *paths) # Соединяет пути с учётом особенностей оп
ерационной системы.
path=os.path.join(os.getcwd(),'untitled0.py')
print(path)#Абсолютный путь до файла 'C:\\Users\\Admin\\untitled0.py
'
os.path.basename(path)#Возвращает имя файла
#>>'untitled0.py'
```

```
os.path.dirname(path)#Возвращает путь до файла
#>>'C:\\Users\\Admin'
os.path.exists(path)#Возвращает True, если path указываетт на сущест
вующую директорию или файл
#>>True
os.path.isabs(path)#Возвращает True, если path является абсолютным
#>>True
os.path.isfile(path)#Возвращает True, если path является файлом
#>>True
os.path.isdir(path)#Возвращает True, если path является директорией
#>>False
os.path.split(path)#Возвращает кортеж (head, tail), где tail –
 все, что содержится после крайнего слеша a head – все остальное.
#>>('C:\\Users\\Admin', 'untitled0.py')
```

Встроенные средства python для работы с файлами: открытие / закрытие, чтение и запись.

Для работы с файлом его надо открыть, используя встроенную функцию open:

```
f = open(file, mode='r', buffering=-1, encoding=None, er-
    rors=None, newline=None, closefd=True, opener=None)
```

где file – абсолютный или относительный путь до файла;

1) аргумент ***mode*** – режим открытия файла (Файл может быть открыт в режиме ‘ r’ - для чтения, в режиме ‘w’ - для записи и в режиме ‘a’ - для добавления в конец, 'b' открытие в двоичном режиме, 't'-открытие в текстовом режиме, режимы могут быть объединены, то есть, к примеру, 'rb' - чтение в двоичном режиме. По умолчанию режим равен 'rt'.);

2) ***buffering*** – необязательный аргумент, используемый для определения политики буферизации;

3) аргумент ***encoding*** определяет имя кодировки символов, например: 'utf-8' или ‘ascii’;

4) аргумент ***errors*** определяет политику обработки ошибок, связанных с кодировкой символов;

5) аргумент ***newline*** определяет порядок работы в режиме поддержки универсального символа перевода строки и может принимать значения None, ‘\n’, ‘\r’ или ‘\r\n’;

6) аргумент ***closefd*** определяет, должен ли в действительности закрываться дескриптор файла при вызове метода close();

аргумент opener определяет базовый дескриптор файла.

Методы:

```
f.close() # Закрывает файл.
f.write(s) # Записывает строку s.
f.read([n]) # Читает из файла до n байтов.
```

Для того, чтобы вывести на экран содержимое файла можно рассмотреть следующий пример:

```
f = open('C:\\Users\\Admin\\untitled1.py','r') # Открыть файл для чтения
content = f.read() # Прочитать содержимое файла в переменную
print(content) # Вывести содержимое на экран
f.close() # Завершить операции с файлом
```

Вероятность возникновения ошибки при работе с файлами наиболее высока: могут возникнуть ошибки чтения – записи, файл может быть удален в процессе записи, жесткий диск может быть переполнен и т. д. Для обработки исключений можно использовать следующий программный код.

```
try:
    f=open('untitled0.py','rt') #Открытие файла в текстовом режиме д
ля чтения
    print(f.read()) #вывод на экран содержимого файла
except Exception:
    print('Ошибка чтения файла') #При вызове исключения будет выведе
но сообщене
else:
    f.close() #закрывает файл
```

Задание

1) Выведите список всех файлов, располагающихся в каталоге `'D:\'`.

2) В каталоге `'D:\'` создайте текстовый файл и запишите в него 10 случайных чисел, каждое из которых меньше 0.5.

3) Разработать программы, которая генерирует входные данные и сохраняет их в файл.

Лабораторная работа № 7

Разработка приложений с применением браузерного графического интерфейса (4 часа)

Цель работы: изучить инструменты создания программ с применением бразурезного графического интерфейса Jupiter notebook

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Jupiter notebook;
5. Microsoft Visual Studio.

Теоретические сведения
Jupyter notebook

Оболочка IPython является полезным интерактивным интерфейсом для языка Python и имеет несколько удобных синтаксических дополнений к нему. Кроме этого, оболочка IPython тесно связана с проектом Jupyter, предоставляющим своеобразный блокнот (текстовый редактор) для браузера, удобный для разработки, совместной работы и использования ресурсов, а также для публикации научных результатов.

Блокнот Jupiter — браузерный графический интерфейс для командной оболочки IPython и богатый набор основанных на нем возможностей динамической визуализации. Помимо выполнения операторов Python/IPython, блокнот позволяет пользователю вставлять форматированный текст, статические и динамические визуализации, математические уравнения и многое другое. Хотя просмотр и редактирование блокнота IPython осуществляется через окно браузера, он должен подключаться к запущенному процессу Python для выполнения кода. Для запуска ядра выполните следующую команду в командной строке вашей операционной системы (рис 6):

jupyter notebook

В качестве рабочей директории будет указан каталог, в котором будет выполнен запуск ядра. Для смены каталога в командной строке можно использовать команду cd. Для завершения работы ядра введете сочетание клавиш Ctrl+C.

Язык программирования Python и его экосистема для исследования данных являются клиентоориентированными, и в значительной степени это проявляется в доступе к документации. Каждый объект Python содержит ссылку на строку, именуемую docstring (сокращение от documentation string — «строка документации»), которая в большинстве случаев будет содержать краткое описание объекта и способ его использования.

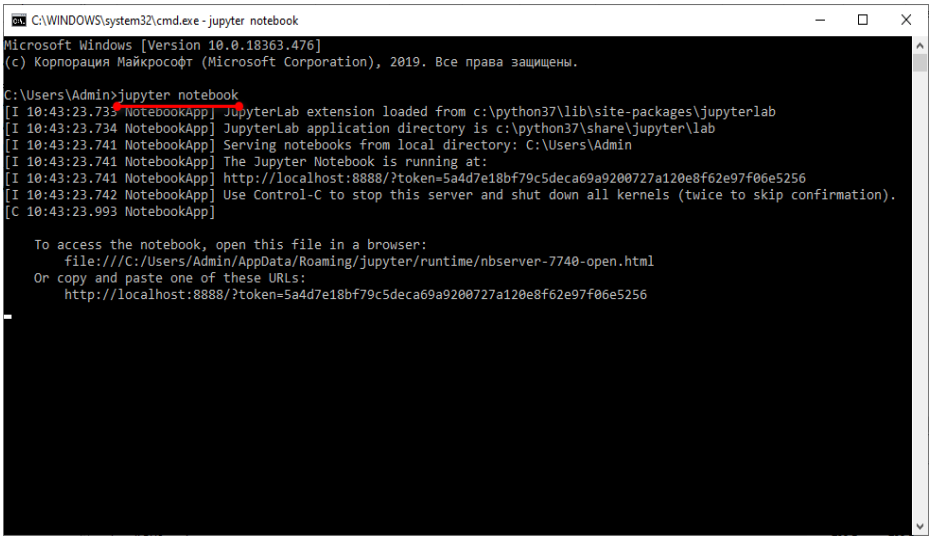


Рис. 6. Запуск jupyter notebook

Доступ к документации

В языке Python имеется встроенная функция help(), позволяющая обращаться к этой информации и выводить результат. Например, чтобы посмотреть документацию по встроенной функции len, можно сделать следующее:

```
help(print)
Help on built-in function print in module builtins:
print(...)
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

В зависимости от интерпретатора информация будет отображена в виде встраиваемого текста или в отдельном всплывающем окне.

Задание

Разработайте программу при помощи интерфейса Jupiter Notebook.

Лабораторная работа № 8

Обработка массивов данных на языке Python
(4 часа)

Цель работы: изучить инструменты создания графиков и диаграмм с применением конструкций языка Python.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Индексация массива: доступ к отдельным элементам

Если вы знакомы с индексацией стандартных списков языка Python, то индексация библиотеки NumPy будет для вас привычной. В одномерном массиве обратиться к i-му (считая с 0) значению можно, указав требуемый

индекс в квадратных скобках, точно так же, как при работе со списками языка Python:

```
x1
x1[0]
```

Для индексации с конца массива можно использовать отрицательные индексы: `x1[-1]`

Обращаться к элементам в многомерном массиве можно с помощью разделенных запятыми кортежей индексов:

```
x2            #array([[3, 5, 2, 4],
              x2[0,0]
```

Математические операции над элементами массива

Если A и B массивы одинакового размера, то над ними можно выполнять различные математические операции. Эти операции выполняются поэлементно, результирующий массив будет совпадать по геометрии с исходными массивами:

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
                A+B  A-B A*B A/B A**B
```

Для применения тригонометрических, логарифмических и других функций используются методы, в качестве аргументов для которых используются массивы:

```
np.sin(A)
np.log(A)
```

Для выполнения матричного умножения используется метод dot():

```
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
  B = np.array([[1,1],[2,2],[3,3],[4,4]])
                np.dot(A,B)
                np.dot(B,A)
```

Задание

1) Создайте две матрицы размером 3x3 заполненных случайными числами;
2) Транспонируйте одну из матриц
3) Выведите результат умножения данных матриц.

Лабораторная работа № 9

Обработка и визуализация данных на языке Python: построение графиков функций (4 часа)

Цель работы: изучить инструменты создания графиков и диаграмм с применением конструкций языка Python.

Технические средства и программное обеспечение:

1. IBM-PC или совместимый компьютер;
2. Операционная система Microsoft Windows;
3. Пакет офисных программ Microsoft Office;
4. Microsoft Visual Studio.

Теоретические сведения

Matplotlib

Matplotlib — мультиплатформенная библиотека для визуализации данных, основанная на массивах библиотеки NumPy и спроектированная в расчете на работу с обширным стеком SciPy. Она была задумана Джоном Хантером в 2002 году и изначально представляла собой патч к оболочке IPython, предназначенный для реализации возможности интерактивного построения с помощью утилиты gnuplot графиков в стиле MATLAB из командной строки IPython. Создатель оболочки IPython Фернандо

Импорт matplotlib

Аналогично тому, как мы использовали сокращение np для библиотеки NumPy и сокращение pd для библиотеки Pandas, мы будем применять стандартные сокращения для импортов библиотеки Matplotlib:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Чаще всего мы будем использовать интерфейс plt.

Простые линейные графики

Вероятно, простейшим из всех графиков является визуализация отдельной функции y = f(x). В этом разделе мы рассмотрим создание простого графика такого типа. Как и во всех следующих разделах, начнем с настройки блокнота для построения графиков и импорта функций, которые будем использовать:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

Во всех графиках Matplotlib мы начинаем с создания рисунка и системы координат. В простейшем случае рисунок и систему координат можно создать следующим образом:

```
fig = plt.figure()
 ax = plt.axes()
```

В библиотеке Matplotlib можно рассматривать рисунок (экземпляр класса plt.Figure) как единый контейнер, содержащий все объекты, представляющие системукоординат, графику, текст и метки. Система координат (она же — оси координат, экземпляры класса plt.Axes) — то, что вы видите выше: ограничивающий прямоугольникс делениями и метками, в котором потом будут находиться составляющие нашу визуализацию элементы графика. В этой книге мы будем использовать имя переменной fig для экземпляра рисунка и ax для экземпляров системы координат или группы экземпляров систем координат. После создания осей можно применить функцию ax.plot для построения графика данных. Начнем с простой синусоиды:

```
fig = plt.figure()
 ax = plt.axes()
x = np.linspace(0, 10, 1000)
 ax.plot(x, np.sin(x));
```

Мы могли бы воспользоваться и интерфейсом pylab, при этом рисунок и система координат были бы созданы в фоновом режиме:

```
plt.plot(x, np.sin(x));
```

Настройка графика: цвета и стили линий

Первое, что вы можете захотеть сделать с графиком, — научиться управлять цветами и стилями линий. Функция plt.plot принимает дополнительные аргументы, которыми можно воспользоваться для этой цели. Для настройки цвета используйте ключевое слово color, которому ставится в соответствие строковый аргумент, задающий практически любой цвет. Задать цвет можно разными способами:

```
plt.plot(x, np.sin(x - 0), color='blue') # Задаем цвет по названию
plt.plot(x, np.sin(x - 1), color='g') # Краткий код цвета (rgbcmyk)
plt.plot(x, np.sin(x -
 2), color='0.75') # Шкала оттенков серого цвета,
# значения в диапазоне от 0 до 1
plt.plot(x, np.sin(x - 3), color='#FFDD44') # 16-ричный код
# (RRGGBB от 00 до FF)
plt.plot(x, np.sin(x -
 4), color=(1.0,0.2,0.3)) # Кортеж RGB, значения 0 и 1
plt.plot(x, np.sin(x -
 5), color='chartreuse'); # Поддерживаются все названия
# цветов HTML
```

Если цвет не задан, библиотека Matplotlib будет автоматически перебирать по циклу набор цветов по умолчанию при наличии на графике нескольких линий.

Стиль линий можно настраивать и с помощью ключевого слова linestyle:

```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
# Можно использовать и следующие сокращенные коды:
plt.plot(x, x + 4, linestyle='-') # сплошная линия
plt.plot(x, x + 5, linestyle='--') # штриховая линия
plt.plot(x, x + 6, linestyle='-.') # штрихпунктирная линия
plt.plot(x, x + 7, linestyle=':'); # пунктирная линия
```

Если вы предпочитаете максимально сжатый синтаксис, можно объединить задание кодов linestyle и color в одном неключевом аргументе функции plt.plot:

```
plt.plot(x, x + 0, '-g') # сплошная линия зеленого цвета
plt.plot(x, x + 1, '--c') # штриховая линия голубого цвета
plt.plot(x, x + 2, '-.k') # штрихпунктирная линия черного цвета
plt.plot(x, x + 3, ':r'); # пунктирная линия красного цвета
```

Эти односимвольные коды цветов отражают стандартные сокращения, принятые в широко используемых для цифровой цветной графики цветовых моделях RGB (Red/Green/Blue — «красный/зеленый/синий») и CMYK (Cyan/Magenta/Yellow/blacK — «голубой/пурпурный/желтый/черный»).

Настройка графика: пределы осей координат

Библиотека Matplotlib достаточно хорошо подбирает пределы осей координат по умолчанию, но иногда требуется более точная настройка. Простейший способ настройки пределов осей координат — методы plt.xlim() и plt.ylim():

```
plt.plot(x, np.sin(x))
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```

Если вам нужно, чтобы оси отображались зеркально, можно указать аргументы в обратном порядке:

```
plt.plot(x, np.sin(x))
plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```

Метки на графиках

В завершение этого раздела рассмотрим маркирование графиков: названия, метки осей координат и простые легенды. Названия и метки осей — простейшие из подобных меток. Существуют методы, позволяющие быстро задать их значения:

```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve") # Синусоидальная кривая
plt.xlabel("x")
plt.ylabel("sin(x)");
```

Гистограммы, разбиения по интервалам и плотность

Простая гистограмма может принести огромную пользу при первичном анализе набора данных:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data);
```

У функции hist() имеется множество параметров для настройки как вычисления, так и отображения. Вот пример гистограммы с детальными пользовательскими настройками:

```
plt.hist(data, bins=30,
         alpha=0.5, histtype='stepfilled',
         color='steelblue', edgecolor='none');
```

Docstring функции plt.hist содержит более подробную информацию о других доступных возможностях пользовательской настройки.

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)
kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```

Задания:

1) Постройте график зависимости частоты вращения шпинделя n от диаметра инструмента d при постоянной скорости резания v. $n = \frac{1000 \cdot v}{\pi \cdot d}$

2) Настройте метки и стиль графика

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Плас Дж. Вандер Python для сложных задач: наука о данных и машинное обучение. — СПб.: Питер, 2018. — 576 с.: ил.
2. Лутц, Марк. Изучаем Python, том 1, 5-е изд.: Пер. с англ. — СПб.: ООО “Диалектика”, 2019. — 832 с. : ил. — Парад, тит. англ.

ОГЛАВЛЕНИЕ

КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКАЯ ИНФОРМАТИКА

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ
для студентов направления 15.03.05 «Конструкторско-технологическое обеспечение машиностроительных производств»
(профили «Технология машиностроения», «Металлообратывающие станки и комплексы», «Конструкторско-технологическое обеспечение кузнечно-штамповочного производства»)
всех форм обучения

Составитель:
Новокщенов Сергей Леонидович

Издается в авторской редакции

Подписано к изданию 30.11.2021.
Уч.-изд. л. 1,9

ФГБОУ ВО «Воронежский государственный технический университет»
394026, Воронеж, Московский просп., 14