Работа №3

«Язык Python»

Цель работы

Получение навыков работы со скриптовыми языками программирования, на примере языка Python

После выполнения работы студенты могут

- Создавать программы на языке Python
- Использовать скриптовые языки для повседневных рутинных задач, не требующих скорости выполнения
- Выбирать язык программирования, исходя из условий разработки

Теоретическая часть

Развитие языков программирования высокого уровня сравнительно быстро привело к существенному отрыву программного кода, написанного программистом, от аппаратной платформы, на которой этот код будет исполняться. На первый план при разработке стали выходить такие факторы, как скорость разработки и удобство сопровождения продукта, так как зачастую стоимость времени программиста и стоимость решения задачи оказываются существенно выше накладных расходов в виде стоимости аппаратного обеспечения. При этом однозначно определился ряд рутинных задач, встающих перед программистом, и требовался набор стандартных быстрых решений для таких задач.

С другой стороны, рост числа программных и аппаратных платформ привел к тому, что использование компилируемых языков программирования стало все более затруднительным. Обеспечить бинарную совместимость между разными платформами очень сложно, и даже бинарная совместимость между разными версиями одной платформы представляет проблему. Поэтому совместимость обеспечивается на уровне исходных кодов, что вынуждает программистов пересобирать свое приложение под разные платформы.

Логичным развитием программного обеспечения стало появление так

называемых **интерпретируемых языков программирования**. В таких языках программный код не компилируется вообще. Он разбирается специальным интерпретатором, который выполняет команды «на лету», по мере прочтения файла с исходными кодами. При этом сам интерпретатор является обычным приложением, однократно собранным для целевой платформы. Программы, написанные на таких языках, называют **сценариями** или **скриптами** (от англ. Script), чтобы отличать их от программ на компилируемых языках.

При использовании интерпретируемых языков достигается сразу несколько преимуществ:

1. С программиста полностью снимается необходимость в изучении нижележащей аппаратной платформы, так как все системные вопросы (в том числе вопросы совместимости и кросс-платформенности) перенесены на уровень интерпретатора.
2. Интерпретатор, в свою очередь, ограничен по числу базовых команд, что существенно упрощает его реализацию, в том числе для разных операционных систем. Сложные операции делаются путем комбинирования простых.
3. Простота интерпретатора позволяет встраивать скриптовые языки внутрь других приложений. В качестве примера можно привести язык JavaScript, сценарии которого выполняются веб браузером на стороне клиента.

Платой за все преимущества является сравнительно низкая скорость работы таких сценариев, так как процесс интерпретации существенно медленнее, чем процесс исполнения готовых машинных команд. Также для исполнения такой программы необходимо наличие программы-интерпретатора. Одним из способов ускорения программ, написанных на таких языках, является так называемая JIT-компиляция (от англ. Just in time), позволяющая перевести сценарий в машинные команды конкретной платформы на этапе загрузки или прямо во время выполнения сценария. Такая компиляция, конечно же, дает худший результат, чем заранее откомпилированная программа, в силу ограничений по времени, и является неким разумным компромиссом между временем загрузки и скоростью работы.

Одними из первых интерпретируемых языков являются языки командных

оболочек. Они позволяют сгруппировать команды, отправляемые в системную командную строку, организовать пользовательский ввод, а также предоставляют базовые средства контроля выполнения, такие как условный переход и цикл. Как правило они используются при решении задач системного администрирования или в качестве установщика программного обеспечения. Одним из первых командных интерпретаторов, поддерживающих сценарии, являлась оболочка Борна (Bourne Shell), используемая в системах семейства Unix. На сегодняшний день активно используется ее расширенная версия под названием Bourne Again Shell или сокращенно BASH. Язык данной оболочки содержит небольшой набор внутренних команд, таких как команды управления выполнением (циклы, условные переходы и т. д.) и команды для перенаправления данных между приложениями. В качестве внешних команд в таком языке может использоваться любое приложение, в том числе и другой сценарий с последовательностью команд такой оболочки.

В системе MS Windows существует аналогичная оболочка — cmd.exe, являющаяся потомком интерпретатора command.com из MS-DOS. Программа на языке такой оболочки оформляется в виде специального пакетного файла с расширением bat. В силу большого упора на графические приложения, в среде MS Windows редко приходится прибегать к помощи командного интерпретатора, в силу этого и язык command.com существенно проще, чем язык других командных оболочек. Идейным продолжателем cmd является оболочка Windows PowerShell.

Большое развитие интерпретируемые языки получили вместе с развитием всемирной паутины. Они используются для создания динамических веб-страниц. Краеугольным камнем здесь является максимальная кросс-платформенность, поэтому при помощи интерпретируемых языков обеспечивается совместимость на уровне исходного кода. При этом отдельно выделяется программный код, исполняемый на стороне пользователя и на стороне сервера. На сервере можно прибегнуть даже к компилируемым языкам программирования для генерации веб-страниц (так называемые CGI-скрипты), однако такая практика в последнее время применяется все реже, так как веб-дизайнеру не следует отвлекаться на системные вопросы.

Одним из наиболее популярных на сегодняшний день интерпретируемых языков программирования является **Python**. Данный язык ориентирован на

повышение производительности разработчика и читаемость исходного кода, и при этом отличается минималистичностью интерпретатора. При разработке Python старались максимально учесть все шероховатости других языков. В то же время, в отличие от других языков программирования, Python обладает колоссальной библиотекой прикладных функций, позволяющей решать большинство возникающих задач. При этом язык отличается легкостью написания модулей. При необходимости можно добавить к языку модуль, вызывающий функции из динамических библиотек, написанных на языках более низкого уровня, таких как C++. Путем переноса «узких мест» программного кода из Python в компилируемый язык, можно добиться хорошей производительности программ, практически не потеряв в их простоте. При этом для языка Python существует большое количество сторонних библиотек. Например, библиотека PyQt позволяет разрабатывать на Python графические приложения с использованием Qt.

Python является языком с открытым исходным кодом и может быть свободно скачан с сайта http://python.org. Там же на сайте присутствует референс-документация для языка. На сегодняшний день существуют две версии Python, поддерживаемые одновременно: Python 2.7 и Python 3. Такое разделение возникло в силу того, что третья версия Python оказалась несовместимой со второй, то есть программа, написанная на Python 2, может не выполниться интерпретатором Python 3. Последняя стабильная вторая версия — Python 2.7 – будет поддерживаться до тех пор, пока не будет полностью произведен переход на Python 3, так как на сегодняшний день существует очень большой объем кода, написанного на Python 2.

Интерпретатор Python может быть написан на многих языках программирования. Его эталонная реализация на языке C – CPython. Простота написания интерпретатора породила много реализаций Python на разных языках, в том числе и на самом Python. Интерпретатор CPython для Windows представляет собой исполняемый файл python.exe, а также набор библиотек на языке Python с небольшим количеством динамических библиотек для «узких мест», требующих максимальной производительности. Дополнительный исполняемый файл pythonw.exe используется для тех случаев, когда нужно выполнить программу на Python без появления на экране текстовой консоли. Полная установка Python 3.6 занимает около 170 мегабайт.

Python является достаточно мощным языком для его использования при математических расчетах. Сторонние библиотеки, такие как NumPy, SciPy и MatPlotLib, позволяют приблизить Python по функциональности к коммерческим математическим пакетам, таким как Matlab и MathCad, а в простых задачах и полностью заменить их. В то же время, интерпретатор Python может работать в интерактивном режиме и использоваться как обычный калькулятор.

Установка сторонних библиотек может производиться несколькими способами:

1. Вручную, скопировав библиотеку в папку lib/site-packages в папке с интерпретатором. Такая установка подходит для библиотек, содержащих только код на языке Python.
2. При помощи установщика, который сам найдет папку Python и скопирует в нее необходимые файлы. Такие установщики могут, в числе прочего, откомпилировать необходимые библиотеки на языках более низкого уровня и установить их в интерпретатор.
3. Используя для установки менеджер пакетов pip, находящуюся в папке Scripts. В качестве аргумента pip может принимать как архив с библиотекой с расширением whl, так и просто название библиотеки. Во втором случае pip автоматически скачает и установит требуемую библиотеку. Pip доступен как в Python 3, так и в Python 2.7 и является рекомендованным способом установки пакетов.

Среды разработки Python

Язык Python, даже в минимальной поставке, содержит средства, достаточные для разработки. Единицей исполнения языка является сценарий (или скрипт), представляющий собой текстовый файл с расширением «.py». Для запуска такого файла необходимо передать его имя в качестве параметра для интерпретатора. Как правило, после установки в среде MS Windows файлы с расширением «.py» автоматически ассоциируются с интерпретатором python.exe, и их можно запускать как обычные исполняемые файлы. Сценарии с расширением «.pyw» ассоциируются с интерпретатором pythonw.exe, то есть запускаются как графические приложения, без текстового терминала.

Редактирование файлов сценариев возможно любым текстовым редактором. Python предоставляет базовую среду разработки — **IDLE**. Для ее запуска необходимо запустить файл «Lib\idlelib\idle.bat» из папки интерпретатора. В систем Windows ярлык для запуска IDLE добавляется в меню «Пуск». Среда представляет собой интерактивную оболочку — такую же, которая доступна при запуске Python из консоли — а также текстовый редактор с подсветкой синтаксиса, из которого можно вызвать сценарий на выполнение.

Помимо этого, существует много различных сред разработки, позволяющих работать с Python. Среди них можно выделить:

- Eclipse с плагином PyDev. Среда поддерживает подсветку синтаксиса, автодополнение, проверку кода на наличие ошибок, а также позволяет запускать сценарии (в том числе в режиме отладки — построчно).
- PyCharm от JetBrains является мощным инструментом профессиональной разработки на Python. Существует как коммерческая, так и бесплатная версия.
- Spyder – бесплатная среда разработки, заточенная на научные вычисления. На основе Spyder построен проект Anaconda, представляющий собой набор современных и востребованных библиотек для научных вычислений, интегрированных со средой разработки, в виде удобного установочного пакета.

Синтаксис языка

Синтаксис языка Python ориентирован на простоту написания и поддержки кода. Язык строго придерживается правила: **одна строка — одна инструкция**. При этом допускается разделять инструкцию на несколько строк, используя символ «\», но не допускается написание нескольких инструкций на одной строке. В случае, если разделение строки произошло внутри скобок, использование символа «\» не требуется.

Используя встроенную функцию `print`, можно вывести любые данные в виде строки. При необходимости Python преобразует вводимые данные. Таким образом, простейшая программа на Python состоит всего из одной строки:

```
print("Hello")
```

Следующие примеры иллюстрируют более сложные варианты записи

инструкций и вывода данных:

```
print\
    ("Hello")
print("This is a very long string, indeed. ",
    "So long, it has to be split apart")
print("sin(1) =", sin(1))
```

Однострочный комментарий в Python определяется при помощи символа «#». Для создания многострочного комментария следует использовать многострочный строковый литерал:

```
# Однострочный комментарий
''' Комментарий
на несколько строк '''
```

Поскольку для Python правило «одна строка — одна инструкция» вынесено на уровень стандарта языка, никакого специального символа, разделяющего отдельные инструкции, не требуется. Однако, запись символа «`;`» в конце строки не приведет к ошибке: Python просто проигнорирует его.

Python является языком с неявной типизацией. Это означает, что переменная может иметь любой тип, который определяется тем, что она в данный момент хранит. Операторы присвоения («=»), проверки присвоенного значения («==»), а также арифметические операторы записываются так же, как в языке C. Точно так же работают операторы «+=» и «-=», а также аналогичные им. Однако существует и возможность множественного присвоения:

```
a, b, c = 10, 12, 15
```

В приведенном примере переменным `a`, `b` и `c` будут присвоены значения 10, 12 и 15 соответственно.

Арифметические операторы в Python полностью аналогичны языку C. Единственным нововведением является оператор «**», обозначающий возведение в степень:

```
print(3 ** 4) # Вывод: 81
```

При этом существует особенность оператора умножения. Python позволяет

умножить массив или строку на целое число. Результатом такой операции является исходный массив или строка, повторенные целое число раз:

```
a = "Hello "*3
print(a) # 'Hello Hello Hello '
```

Переменные и типы данных

Python не требует явного объявления переменной. Переменная появляется в тот момент, когда производится присвоение неизвестному ранее идентификатору. В этот же момент и определяется тип хранимых данных для переменной. При этом допускается присвоение данных другого типа той же переменной: переменная может поменять тип хранимых данных. При необходимости можно удалить переменную, используя оператор del:

```
v = 12                    # Появление целочисленной переменной
v = "Hello world"         # Присвоение строки, смена типа
print(v)                  # Вывод на экран
del v                     # Удаление переменной
```

Тем не менее, существующая переменная всегда имеет определенный тип, и некоторые операции доступны только с определенными типами данных. Например, нельзя произвести операцию деления строки на строку.

Язык Python поддерживает следующие типы данных:

- Булевый тип, используемый в логических вычислениях. Переменная такого типа может принимать одно из двух возможных значений: `True` или `False`.
- Численный тип. Python различает целые и вещественные числа, соответствующие типам `int` и `float`, однако свободно преобразует эти типы между собой. Программист может наткнуться на ограничения только в некоторых специфических случаях. Например, операции битового сдвига работают только с целыми числами. Численные литералы записываются точно так же, как в языке C:
  - `1.15` – вещественное число с плавающей запятой,
  - `114` — обычное целое число,

- `0133` — целое число в восьмеричной системе счисления,
- `0x0AFF` – целое число в шестнадцатеричной системе счисления,
- `6.2e-9` – экспоненциальная форма записи числа, соответствует числу $6.2 \cdot 10^{-9}$.

Дополнительно Python поддерживает работу с комплексными числами. Для записи комплексного числа в конце числового литерала следует добавить букву «j» без пробела:

```
a = 1 + 2j
```

- Строковые данные. Для записи строковых данных существуют строковые литералы. Они записываются в одинарных или двойных кавычках. При этом внутри одинарных кавычек можно свободно использовать двойные и наоборот. Для того, чтобы записать строку в кодировке Unicode, необходимо перед литералом поставить букву «u» без пробела:

```
str = 'String says "Hello"!'
str2 = "It's another string"
ustr = u"Это строка в кодировке Unicode"
```

Для создания строки, которая не помещается в одну строку текстового редактора, следует использовать **тройные** одинарные или двойные кавычки. Такой прием можно использовать как многострочный комментарий:

```
multistr = u"""Это очень длинная строка, которая не помещается
в одну строку текстового редактора и разнесена на несколько
строк для удобства чтения"""
```

Строки можно читать поэлементно, используя оператор квадратных скобок:

```
print(u"Третий символ строки str =", str[3])
```

Однако, строки нельзя изменять. Вместо этого следует создать новую строку. При этом одна строка может быть получена из нескольких путем их склеивания, используя оператор «+», а любые другие данные могут быть преобразованы в строку, используя конструкцию `str()`:

```
s = "Hello, pi = " + str(3.1416)
```

Python позволяет преобразовывать типы между собой используя конструктор

типа. Например, для того, чтобы преобразовать что либо в строку, необходимо использовать следующую форму записи:

```
i = 34
s = str(i)    #  "34"
```

Аналогичным образом происходит преобразование в число:

```
s = "441"
i = int(s)    # 441
f = float(s)  # 441.0
```

Основные операции

Python поддерживает большинство операций, знакомых из языка Си, однако добавляет и часть новых:

- = – Присвоение
- == – Строгое равенство
- <, >, <=, >= – Строгие и нестрогие неравенства
- +, -, *, / – Арифметические операции
- % – Остаток от деления
- ** – Возведение в степень
- <<, >> – Операции битового сдвига
- &, |, !, ^ – Побитовые операции «И», «ИЛИ», «НЕ», «Исключающее ИЛИ»
- and, or, not – Логические операции «И», «ИЛИ», «НЕ»
- in – операция проверки на вхождение элемента в множество:

Также доступна сокращенная форма записи для арифметических операторов. Запись вида:

```
a = a + 4
```

эквивалентна записи:

```
a += 4
```

Составные типы данных

Помимо простых типов данных, Python позволяет группировать данные в списки, словари и тьюплы. Рассмотрим их подробнее.

Тьюпл представляет собой неизменяемый набор данных. Элементами тьюпла могут быть совершенно разнородные типы данных, в том числе другие тьюплы или списки. В тьюпле элементы нумеруются с нуля, так же как и переменные в массивах языка C. Особенность Python состоит в том, что имеют смысл и отрицательные номера элемента: они используются для нумерации с конца. Элемент с номером «-1» — это последний элемент тьюпла, «-2» — предпоследний и так далее. Для определения тьюпла следует указать несколько элементов через запятую, заключив их в круглые скобки:

```
tuple1 = (1, '2', "string", (2, 'Where the hell am I?'), 6)
```

Доступ к элементам тьюпла осуществляется при помощи оператора квадратных скобок:

```
first = tuple1[0]
last  = tuple1[-1]
```

При необходимости можно выбрать несколько идущих подряд элементов из тьюпла, указав в квадратных скобках два числа, разделенных двоеточием, при этом будут выбраны все элементы, включая первый и не включая последний. Нижеприведенный код сохранит в `sub` элементы со 2 по 4 включительно:

```
sub = tuple1[2:5]
```

Для определения размера тьюпла следует использовать стандартную функцию `len`:

```
tsize = len(tuple1)
```

Для создания тьюпла, состоящего из одной переменной, необходимо записать в тьюпл эту переменную и поставить после нее запятую:

```
t = (22,)
```

Для создания пустого тьюпла следует использовать пустые круглые скобки, либо встроенную функцию `tuple()`:

```
empty_t = ()
```

```
empty_t = tuple()
```

Список представляет собой массив каких-либо элементов, объединенных общим названием и доступных по индексу. Индексы в списке имеют тот же смысл, что и в тьюпле, и список поддерживает все те же операции, что и тьюпл, но, в отличие от тьюпла, элементы списка можно изменять, а также добавлять и удалять элементы в произвольном месте списка. Список определяется так же, как и тьюпл, однако используются квадратные скобки:

```
ar = [1, "2", '3', "Ololo", 4]
```

Добавление элементов в конец списка происходит при помощи метода `append`, для добавления в середину существует метод `insert`. Удаление элементов производится при помощи метода `pop`. Пример ниже показывает, как можно модифицировать список:

```
ar = [1, "2", '3', "Ololo", 4]
ar[1] = 15            # Изменить элемент под номером 1
ar.insert(0, 10)      # Добавить число 10 перед нулевым элементом
ar.append("Last one") # Добавить строку "Last one" в конец списка
ar.pop(3)             # Убрать элемент под номером 3
```

Список может содержать всего один элемент или может вовсе быть пустым. При необходимости можно создать список необходимой длины, используя оператор умножения. Следующий код создаст массив из 100 нулевых элементов:

```
empty = [0] * 100
```

Словарь представляет собой список, в котором элементы идентифицируются не по индексу, а по переменной-ключу. В качестве ключа может выступать любой тип данных, поддерживаемый оператором сравнения, но на практике используются строки или номера, идущие не по порядку. Словарь определяется как набор пар «КЛЮЧ : ЗНАЧЕНИЕ», разделенных запятыми, которые заключены в фигурные скобки:

```
rec = {"one": 1, "two":2}
```

Обращение к словарю происходит при помощи оператора квадратных скобок, в которые заключен ключ:

```
b = rec["one"]
```

Присвоение значения элементу словаря происходит таким же образом. При этом если ключ не существует в словаре, будет создан новый элемент с указанным ключом:

```
rec["three"] = 3 # Создать новый элемент "three":3
```

Для того, чтобы проверить, существует ли элемент с заданным ключом в словаре, существует оператор in:

```
b = ("one" in rec) # b == True если такой элемент есть в словаре
```

Ввод и вывод

Текстовый ввод и вывод в Python реализованы стандартными средствами, используя функции `input` и `print`.

При выводе через `print` язык автоматически преобразует типы данных, позволяя выводить в текстовом виде даже сложные структуры.

Ввод данных всегда происходит в виде строки, однако строка может быть преобразована к целому числу при помощи функции `int()`, либо к вещественному при помощи функции `float()`:

```
a = int(input("Введите число"))
print("Вы ввели ", a)
```

Операторы управления выполнением

Управление выполнением в Python реализовано в общих чертах так же, как и в других структурных языках программирования. Особенностью Python является оформление операторных блоков. В классических языках программирования специально выделяются операторы, ограничивающие операторный блок. В языке C и подобных ему по синтаксису это фигурные скобки, «`{}`». При этом устоявшейся практикой является выделение операторного блока отступом от 2 до 8 пробелов, чтобы улучшить читаемость кода. В языке Python отступ является **требованием** при оформлении операторного блока, а специальные операторы отсутствуют полностью. Размер отступа не регламентирован, но должен быть одинаковым у всего блока. Строка, требующая после себя операторный блок с увеличенным отступом, содержит в конце символ двоеточия. Покажем это на примере **оператора условного перехода**:

```
a = int(input());
if a > 0:
    print("a is bigger than 0")
else:
    print("a is below 0")
print("Done")
```

Работа рассмотренного выше оператора `if` аналогична языку C. Основная ветвь условного перехода будет выполнена, если выражение после `if` истинно (то есть, результат его вычисления равен `True`), либо если полученное в результате вычисления число не равно 0, либо если в качестве условия в `if` передана непустая строка. В противном случае (`False`, 0 или пустая строка) будет выполнена альтернативная ветвь, начинающаяся со слова `else`, если она присутствует. После выполнения условия, сценарий продолжит исполняться с первой строки, отступ которой равен отступу оператора `if`.

Условия в операторе `if` можно группировать при помощи скобок, а также комбинировать, используя операторы логического И (`and`), логического ИЛИ (`or`), а также логического отрицания (`not`). При этом доступны и более мощные инструменты:

- Двойные сравнения позволяют, в частности, проверить переменную на принадлежность к диапазону:

```
if 0 < a < 10:
    print("a is in (0;10)")
```

- Оператор `in` позволяет определить, входит ли переменная в заданный набор значений:

```
tpl = (1, 2, 3, 5, 7)
if a in tpl:
    print("a is simple")
if a in tpl[1:]:
    print("a is simple and not 1")
```

Оператор `in` работает для любого набора значений, в том числе и для строк:

```
s = "Software says 'hello' to you"
b = "hell" in s                          # True
```

Для задания **цикла** используется один из двух операторов: `while` или `for`. По своей логике работы они полностью аналогичны языку C. Оператор `while` продолжает выполнение тела цикла до тех пор, пока выполняется условие, записанное после него. Если условие не выполняется, то следующей будет выполнена строка, отступ которой равен отступу оператора `while`:

```
a = 0
while a<10:
    print(a)
    a += 1
print("Done")
```

Оператор `for` служит для запуска цикла по заданному набору значений. При этом при задании цикла `for` обязательно использование оператора `in`. В качестве набора значений можно использовать любое множество. Например, тьюпл, список или даже строку:

```
# Вывод букв в строке по одной
for a in "Hello world":
    print(a)
```

Для задания арифметической прогрессии можно использовать встроенный оператор `range`, задающий диапазон целых чисел. Возможен один из трех вариантов использования range:

- `range(start, end, step)` – создать набор значений от `start` до `end`, не включая последний, с шагом `step`
- `range(start, end)` – создать набор значений от `start` до `end`, не включая последний, с шагом 1.
- `range(end)` – создать набор значений от 0 до `end` с шагом 1, не включая `end`.

Использование `range` вместе с `for` позволяет определить цикл по числовой последовательности, например по индексам элементов списка:

```
# Вывод чисел от 1 до 9
for a in range(1, 10):
    print(a)
```

Внутри циклов можно использовать операторы `break` и `continue`:

- Оператор `break` прерывает текущий цикл полностью и сразу переходит к выполнению первой строки после операторного блока цикла.
- Оператор `continue` немедленно переходит на начало следующей итерации цикла, пропуская оставшиеся строки до конца операторного блока

```
for a in range(1, 100)
if a % 10 == 0:
    continue # Пропуск итераций, в которых a делится на 10
if a>50:
    break # Досрочный выход из цикла после 50 итераций
```

Наконец, существует специальный оператор `pass`. Он не делает ничего и служит только для того, чтобы обозначить наличие пустого операторного блока:

```
for a in range(1, 100):
    if(a == 15):
        # print(a)
        pass;
```

Функции и модули

Для задания **функции** используется ключевое слово `def`. Аргументы функции указываются в круглых скобках после названия функции, затем указывается двоеточие и описывается тело функции в виде операторного блока (с отступом). Тип возвращаемого значения, в отличие от языка C, не указывается. Функция может вернуть любое значение, простое или сложное, используя оператор `return`, либо не вернуть ничего (`None`).

Для примера, создадим функцию, вычисляющую сумму натуральных чисел меньше заданного:

```
def N_sum(x):
    if x <= 0:
        print("Error: x must be positive")
        return 0;

    a = 1;
    res = 0;
    while a<x:
        res += a;
        a += 1;
    return res
```

Для вызова функции следует написать ее название и поставить круглые скобки. В случае, если функция принимает какие-то параметры, их также следует указать в круглых скобках. Результат выполнения функции (переменную, возвращенную из функции ключевым словом `return`) можно присвоить какой-либо переменной:

```
n = N_sum(10)
print("n = ", n)
```

Можно задать в функции значения по умолчанию для аргументов:

```
def func(a, b=10):
    return a*b
```

Тогда при вызове функции не обязательно указывать все параметры:

```
print(func(3))  # Вывод: 30
```

В силу того, что программа на языке Python разбирается построчно, функция должна быть определена в файле выше, чем она будет вызвана. При этом в Python нельзя создать прототип (объявление) функции. Также важной особенностью Python является отсутствие какой-либо специальной "точки входа", аналогично функции `main` в языке C. Инструкции, составляющие программу, могут быть записаны подряд без какого-либо оформления. Выполнение программы начинается с первой строки без отступа.

Наконец, для того, чтобы подключить какой-либо сторонний **модуль**,

используется ключевое слово `import` в сочетании со словом `from`. При помощи `import` можно импортировать весь модуль целиком, используя символ «*», либо импортировать только отдельные функции или классы. К примеру, можно следующим образом импортировать библиотеку математических функций, а также из библиотеки комплексных функций импортировать функцию вычисления фазы:

```
from math import *
from cmath import phase
```

При необходимости можно импортировать внешнюю функцию или объект и присвоить ему короткое имя, используя ключевое слово `as`. Пример иллюстрирует, как можно импортировать класс для построения графиков из `matplotlib` под именем `pp`:

```
import matplotlib.pyplot as pp
```

Стандартная библиотека Python содержит большое количество модулей, которые можно импортировать. Полный список модулей, а также экспортируемых из них функций и классов доступен по адресу: https://docs.python.org/3/library/index.html

Документация на сторонние программные модули предоставляется вместе с модулями. Для ознакомления с документацией на библиотеку matplotlib следует обратиться по адресу http://matplotlib.org/1.4.3/users/beginner.html.

Использование Qt в Python

В языке Python также можно использовать возможности Qt. Для этого необходимо установить библиотеку PyQt. Сделать это можно при помощи pip, выполнив команду в папке Scripts интерпретатора Python:

```
pip install PyQt5 pyqt5-tools
```

Можно также скачать установочные файлы PyQt в виде установочного пакета на сайте http://sourceforge.net/projects/pyqt/ или https://www.riverbankcomputing.com/software/pyqt/intro.

После установки в Python появится программный модуль PyQt и подмодули, аналогичные модулям Qt. Например, модуль PyQt5.QtWidgets соответствует модулю Widgets из Qt, подключаемому в проектный файл командой

«QT += widgets». При этом строки QString можно не использовать, так как классы PyQt используют строки Python: их функционала вполне достаточно.

К сожалению, использование Qt Creator для написания программ с использованием PyQt невозможно. Тем не менее, для разработки интерфейса приложения доступна программа Qt Designer, при помощи которой можно создать файл ui с описанием интерфейса. Создание файла ui полностью аналогично тому, как это происходит в Qt Creator.

После того, как файл ui создан, необходимо загрузить его в python-скрипт и описать логику его работы. Скрипт, который загружает файл ui и выводит на экран окно, при использовании Qt5, представлен ниже:

```
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5 import uic

app = QApplication([])
ui = uic.loadUi("PyQtForm.ui")
ui.show()
exit(app.exec())
```

В случае использования Qt4, код немного меняется:

```
from PyQt4.QtGui import QApplication, QWidget
from PyQt4 import uic

app = QApplication([])
ui = uic.loadUi("PyQtForm.ui")
ui.show()
exit(app.exec_())
```

Объекты, созданные в дизайнере, будут доступны внутри объекта ui под своими именами. Например, для смены надписи на кнопке с именем pushButton необходимо записать:

```
ui.pushButton.setText("Hello")
```

Для того, чтобы обеспечить логику работы интерфейса, необходимо

написать ряд функций, обрабатывающих возникающие события. В Python нет механизма сигналов и слотов, поэтому в PyQt сигналы Qt преобразуются в специальные классы, которые могут быть подключены к обычным функциям. Описание слота, который при нажатии на кнопку изменяет написанный на ней текст, приведено в листинге:

```
def onClick():
    s = w.pushButton.text()
    w.pushButton.setText("_" + s + "_")
```

Для соединения данного сигнала со слотом необходимо использовать метод `connect` соответствующего сигнала, которому в качестве аргумента передается название объявленной функции. Для соединения сигнала `clicked` объекта `pushButton` с созданной функцией следует записать следующий код:

```
w.pushButton.clicked.connect(onClick)
```

После выполнения данной строки любое нажатие на кнопку `pushButton`, порождающее сигнал `clicked`, будет вызывать выполнение кода, записанного в функции `onClick`.

Аналогичным образом можно соединить сигнал одного виджета со слотом другого. К примеру, следующим образом можно соединить сигнал `valueChanged` одного слайдера со слотом `setValue` другого слайдера, чтобы их значения изменялись синхронно:

```
w.horizontalSlider.valueChanged.connect(w.verticalSlider.setValue)
```

PyQt поддерживает весь функционал библиотеки Qt, транслируя его в Python, за редкими исключениями. Например, вместо `QString` почти всегда можно использовать обычные строки Python. Также, в Python недоступна форма записи «`::`», используемая в C++. При необходимости использовать константы Qt необходимо импортировать модуль `Qt` из модуля `QtCore`:

```
from PyQt5.QtCore import Qt  # Версия для PyQt5
from PyQt4.QtCore import Qt  # Версия для PyQt4
```

При необходимости использовать константу `Qt::AlignCenter` необходимо записать в Python «`Qt.AlignCenter`». Аналогичным образом

следует поступать с остальными константами из пространства имен `Qt`.

Документация по классам Qt доступна онлайн https://doc.qt.io/qt-5/reference-overview.html (на английском языке), http://doc.crossplatform.ru/qt/4.8.x/html-qt/ (на русском языке для версии Qt 4.8, основной функционал не отличается от Qt5) и в программе Qt Assistant, устанавливаемой вместе с библиотекой.

Порядок проведения работы

В данной работе необходимо написать скрипт на языке Python, который моделирует работу цифрового фильтра. Фильтр реализует поведение дифференцирующей или интегрирующей RC-цепи. Для расчета цифрового сигнала на выходе цепи необходимо знать отсчеты цифрового сигнала на входе, а также цифровую импульсную характеристику цепи, отсчеты которой вычисляются следующим образом:

- $h[n]=\frac{1}{\tau}\exp(-\frac{n}{\tau})$ для интегрирующей цепи, где τ - постоянная времени цепи, деленная на интервал дискретизации
- $h[n]=\begin{cases}1, \text{если } n=0\\ \frac{1}{\tau}(1-\exp(-\frac{n}{\tau})), \text{если } n\neq 0\end{cases}$ для дифференцирующей цепи

Отсчеты импульсной характеристики являются коэффициентами цифрового фильтра.

Зная отсчеты входного сигнала «x», а также отсчеты импульсной характеристики «h», Сигнал на выходе фильтра рассчитывается на основе сигнала на входе при помощи следующего соотношения:

$$y[n]=\sum_{i=0}^{N} h[i]x[n-i]$$, где

y[n] – одиночный отсчет выходного сигнала в момент времени n;

h[i], x[n-i] – отсчеты импульсной характеристики и входного сигнала в соответствующие моменты времени;

N — количество отсчетов импульсной характеристики. Значение N выбирается с учетом соотношения $N<<\tau$.

Для расчета выходного сигнала необходимо повторить расчет y[n] для всего диапазона значений n. Следует исходить из того, что длительность выходного сигнала равна длительности входного.

Для выполнения работы необходимо сделать следующее:

1. Создать основу скрипта

1.1. Создать в среде разработки новый пустой проект на языке Python.

1.2. Импортировать библиотеку для математических вычислений – `math`.

1.3. Импортировать `pyplot` из `matplotlib`.

1.4. Проверить функционирование `matplotlib`, построив произвольный график. Для этого необходимо создать список или тьюпл, содержащий последовательно точки графика, и передать его в функцию `pyplot.plot()`, после чего вызвать функцию `pyplot.show()`, чтобы отобразить график. Например:

```
x = (0, 1, 2, 4, 8, 16, 32, 64)
pyplot.plot(x)
pyplot.show()
```

2. Создать цифровой фильтр

2.1. Создать список, содержащий отсчеты сигнала на входе. В качестве входного сигнала рекомендуется задать импульс единичной амплитуды и произвольной длительности. Для задания сигнала следует использовать цикл.

2.2. Создать функцию, производящую расчет выходного сигнала. Параметром функции должен являться список с отсчетами сигнала на входе (x), а возвращаемым значением должен быть список с отсчетами сигнала на выходе (y), равный по длине входному. Данная функция должна использовать в качестве отсчетов импульсной характеристики следующий тьюпл, обеспечивающий задержку сигнала на 10 тактов:

```
coeff = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1)
```

Необходимо учесть при расчете возможный выход индексов за границы массивов и обработать данную ситуацию.

2.3. Получить отсчеты сигнала на выходе фильтра путем вызова созданной функции.

2.4. Построить графики входного и выходного сигнала с использованием `pyplot` из `matplotlib`. Для того, чтобы построить две кривые на одном

графике, следует два раза вызвать `plot()` и один раз — `show()`.

3. Реализовать при помощи фильтра интегрирующую и дифференцирующую цепь

   3.1. Определить функцию, рассчитывающую импульсную характеристику интегрирующего звена. Функция в качестве аргументов должна принимать постоянную времени цепи и общую длину возвращаемого массива (второму параметру следует задать значение по умолчанию, равное $5 \cdot \tau$). Возвращаемое значение функции — список с отсчетами импульсной характеристики.

   3.2. Модифицировать функцию расчета выходного сигнала таким образом, чтобы в качестве импульсной характеристики использовался результат работы функции, созданной в пункте 3.1. Рекомендуется передавать импульсную характеристику как второй аргумент функции. Для определения длины передаваемого списка следует использовать функцию `len()`.

   3.3. Построить графики входного и выходного сигнала и убедиться, что фильтр моделирует интегрирующее звено.

   3.4. Определить дополнительную функцию, рассчитывающую импульсную характеристику дифференцирующего звена. Повторить моделирование фильтра с использованием данной функции и показать на графике, что фильтр реализует дифференцирующую цепь.

4. **Дополнительное задание.** Обеспечить приложение графическим интерфейсом.

   4.1. При помощи Qt Designer создать графический интерфейс (ui), содержащий следующие компоненты:

      4.1.1. SpinBox для задания количества отсчетов во входном и выходном сигнале,

      4.1.2. DoubleSpinBox для задания постоянной времени цепи,

      4.1.3. Два RadioButton'a для выбора типа цепи — дифференцирующей или интегрирующей.

      4.1.4. Кнопку PushButton с надписью «Построить».

   4.2. Добавить создание и вызов интерфейса.

4.3. Создать функцию, обрабатывающую нажатие на кнопку, и соединить ее с сигналом `click` от кнопки.

4.4. Внутри функции обработки нажатия на кнопку следует получить из интерфейса тип фильтра и постоянную времени, создать массив, хранящий отсчеты импульсной характеристики фильтра, сгенерировать входной сигнал в виде прямоугольного импульса и рассчитать выходной сигнал. Два полученных сигнала следует вывести на график с помощью matplotlib.

4.5. Запустить приложение и убедиться, что изменение параметров фильтра отражается на графиках.

4.6. При возникновении конфликтов между `matplotlib` и `PyQt` необходимо в начале файла, до импорта `pyplot`, записать:

```
import matplotlib
matplotlib.use('Qt5Agg')
```

При использовании PyQt4 необходимо заменить `Qt5Agg` на `Qt4Agg`.

Вопросы для самоконтроля и подготовке к защите работы №3

1. Чем отличаются интерпретируемые языки программирования от компилируемых?
2. Как в Python можно вывести на экран текстового терминала значение переменной?
3. Какие составные типы данных в Python вы знаете?
4. Как в Python определяется операторный блок?
5. Какие существуют способы задания цикла в Python?
6. Как объявить функцию? Где в файле скрипта может располагаться определение функции?
7. * Как обработать сигнал от объекта в PyQt?