Технология программирования

Среда разработки программ. Структура программы. Типы данных и переменные. Операции и встроенные функции

Вопрос 1. Среда разработки. Структура программы. Оформление исходного кода программы на языке программирования высокого уровня

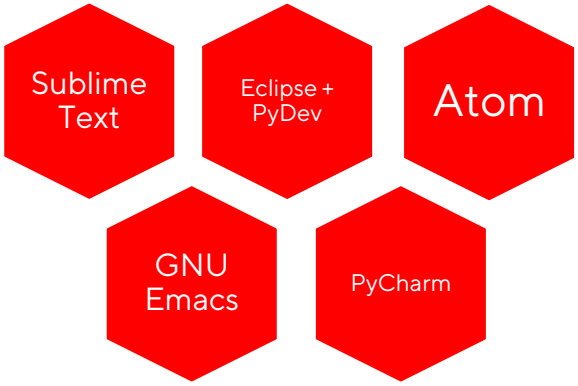
Среда разработки (IDE) — это программное обеспечение, которое объединяет в одном месте все нужные инструменты для разработки программного обеспечения. С ней писать, тестировать и отлаживать код удобнее и эффективнее.

Основные компоненты IDE

1. Текстовый редактор.
2. Компилятор или интерпретатор.
3. Отладчик.
4. Система управления версиями.
5. Графический интерфейс пользователя (GUI).
6. Плагины и расширения.
7. Интеграция с другими инструментами и сервисами.

Схема 1. Среда разработки с поддержкой Python

Среда разработки IDE с поддержкой Python



Элементы структуры программы на Python

1. Импорт библиотек — здесь вы можете определить константы, которые будут использоваться в программе.
2. Константы — на данном этапе определяются постоянные величины.
3. Функции — в этой секции определяются функции, что помогает структурировать код и уменьшить дублирование.
4. Основной код — здесь расположена логика программы.
5. Обработка ошибок (опционально) — здесь можно добавить обработку исключений, чтобы управлять возможными ошибками выполнения.

Эта структура считается хорошим стартом, чтобы создать программу на Python. Ее можно дополнять или изменять в зависимости от потребностей проекта.

Синтаксис Python отличается от других языков программирования, и его главная особенность заключается в способе выделять блоки кода. Python представляет собой регистронезависимый язык. Это означает, что буквы в разных регистрах считаются разными символами. Например, 't' и 'T' — это не одно и то же.

Комментарии в коде позволяют разработчикам оставлять заметки, которые не будут интерпретироваться интерпретатором, но могут служить полезными подсказками. Чтобы прокомментировать одиночную строку, нужно просто поставить символ # в начале строки. Хотя стандартного способа создавать многострочные комментарии в Python нет, можно обойти это ограничение и заключить текст в тройные кавычки (""").

Чтобы в Python обозначить блоки, используется отступ, или пробел. Строки, у которых одинаковый отступ, относятся к одному блоку кода. Рекомендуется использовать четыре пробела для создания отступов, хотя в некоторых специальных редакторах можно применить табуляцию.

Важно помнить, что отступы недопустимы для строк, которые не вложены, так как это приведет к синтаксической ошибке. В**ложенные строки — это** строки кода, которые находятся внутри определенного блока, такого как функция, цикл, условие, или любая другая конструкция, которая требует отступов для обозначения уровня вложенности.

```
x = 1; y = 2; z = x + y # это правильная запись
 a = 2 b = 3 c = a + b # так писать нельзя
print(z) # правильная запись
Print(c) # так писать нельзя, с большой буквы Print неизвестен # интерпретатору
""" многострочный
        комментарий """
```

Вопрос 2. Редактирование простых макросов

В каждой профессии рано или поздно наступает момент, когда нужно выполнять задачи на компьютере, которые повторяются. Автоматизировать этот процесс просто, так как алгоритм очевиден, но на практике найти подходящие инструменты для автоматизации оказывается сложнее.

В этом контексте рассмотрим, как создать макросы с помощью языка программирования Python. Они записывают, а затем воспроизводят действия с помощью мыши и клавиатуры. Управлять такой программой надо будет через графический интерфейс. Сначала понадобится библиотека, которая фиксирует действия мыши и клавиатуры. Надо выбрать библиотеку Pynput.

Чтобы установить библиотеку Pynput в Python, следует выполнить следующие шаги:

- открыть командную строку или терминал;
- ввести команду pip install pynput и нажать Enter;
- дождаться, пока завершится установка;
- после успешной установки появится сообщение об этом.

Это позволит вам начать использовать Pynput в ваших проектах на Python.

Давайте перейдем к практике и изучим, как это реализовать.

Надо создать автоматический кликер с помощью Python. Код принимает ввод с клавиатуры, когда пользователь нажимает клавишу start, и завершает автокликер, когда пользователь нажимает клавишу exit.

Автокликер начинает щелкать везде, где указатель находится на экране.

Ход работы

1. Убедитесь, что модуль Pynput был успешно установлен в вашу рабочую среду. Для этого откройте программу IDLE, которая используется для работы с кодом на языке Python. Введите команду import pynput. После выполнения этого на выходе должно быть ноль ошибок, что означает, что модуль успешно установлен.

2. Теперь подготовьте код к созданию автоматизированного скрипта. Импортируйте время, чтобы установить задержки между щелчками мыши, и потоки, которые позволяют выполнять несколько действий параллельно. А затем — кнопку, чтобы управлять кнопкой мыши, и контроллер, чтобы контролировать положения и действия мыши, из модуля **pynput.mouse**. Импортируйте прослушиватель, чтобы отслеживать нажатия клавиш, и код ключа, чтобы идентифицировать нужные клавиши, из **pynput.keyboard.**

```
# importing time and threading
import time
import threading
from pynput.mouse import Button, Controller

# pynput.keyboard is used to watch events of
# keyboard for start and stop of auto-clicker
from pynput.keyboard import Listener, KeyCode
```

3. Создайте четыре переменные:

    - **задержка** между каждым щелчком мыши (в секундах);
    - **кнопка** используется, чтобы нажимать в любом направлении, в котором вы хотите;
    - **start_stop_key**: клавиша используется, чтобы запустить и остановить клик при запуске программы, когда надо выполнить автокликер. Он должен быть из ключевого класса или устанавливаться с помощью ключевого кода;
    - **exit_key**: ключ, используемый для завершения выполняемого автокликера. Он должен быть из класса key или задан с помощью KeyCode.
    -

```
# four variables are created to
# control the auto-clicker
delay = 0.001
button = Button.right
start_stop_key = KeyCode(char='a')
stop_key = KeyCode(char='b')
```

4. Создайте класс, который будет основан на потоке (threading.Thread в Python), чтобы действия автокликера выполнялись параллельно с другими задачами программы. Потоки позволяют автокликеру работать независимо, так, что он не блокирует основную программу. Передайте delay и button классу с двумя флагами, которые проверяют, запущен ли автокликер, и выключен ли он полностью. Так можно проверить, выполнилась программа или нет.

```
# threading.Thread is used
# to control clicks
class ClickMouse(threading.Thread):

 # delay and button is passed in class
 # to check execution of auto-clicker
  def __init__(self, delay, button):
      super(ClickMouse, self).__init__()
      self.delay = delay
      self.button = button
      self.running = False
      self.program_running = True
```

5. Добавьте методы для внешнего управления потоками.

```
class ClickMouse:
   def start_clicking(self):
      self.running = True

   def stop_clicking(self):
      self.running = False

   def exit(self):
      self.stop_clicking()
      self.program_running = False
```

6. При запуске потока создайте метод program_running выполняется в цикле до тех пор, пока значение не примет значение true. А также создается другой цикл внутри существующего цикла, где он проверяет, установили ли для running значение true или нет. В случае, если мы находимся внутри обоих циклов, он нажмет на кнопку set и перейдет в режим ожидания для заданной задержки.

```
# method to check and run loop until
# it is true another loop will check
# if it is set to true or not,
# for mouse click it set to button
# and delay.
def run(self):
      while self.program_running:  # Внешний цикл
         while self.running:  # Вложенный цикл
            mouse.click(self.button)  # Клик мышью
            time.sleep(self.delay)  # Задержка между кликами
         time.sleep(0.1)  # # Короткая задержка для проверки флагов
```

7. Создайте экземпляр для контроллера мыши, затем создайте поток ClickMouse. Запустите экземпляр, чтобы перейти в цикл внутри метода run.

```
# instance of mouse controller is created
mouse = Controller()
click_thread = ClickMouse(delay, button)
click_thread.start()
```

8. Создайте метод on_press, который принимает клавишу в качестве аргумента и настраивает прослушиватель клавиатуры. Клавиша start_stop_key совпадает с клавишей start(a) при ее выполнении. Затем щелчок прекращается, когда флаг running установлен в True в потоке. Метод Exit вызывается в методе, если клавиша exit(b) выполняется и останавливает прослушиватель.

```
from pynput.keyboard import Listener, KeyCode

# Метод для обработки нажатий клавиш
def on_press(key):
    # Клавиша `start_stop_key` запускает/останавливает автокликер
    if key == start_stop_key:
        if click_thread.running:
            click_thread.stop_clicking()
        else:
            click_thread.start_clicking()
    # Клавиша `stop_key` завершает автокликер
    elif key == stop_key:
        click_thread.exit()
        listener.stop()

# Настройка прослушивателя клавиатуры
with Listener(on_press=on_press) as listener:
    listener.join()
```

После запуска кода можно видеть в выходных данных количество нажатий, которые сделал автокликер. Он совместим с Windows, Mac и Linux.

Автокликер — полезное программное обеспечение для систем, поскольку позволяет экономить разумное количество времени, которое затрачивается на многократное нажатие.

Вопрос 3. Типы данных. Ключевые слова. Стандартные типы данных и действия с ними. Преобразования типов. Объявление переменных. Выражения. Правила построения выражений. Работа с числовыми данными

Типы данных в Python — это классификация данных, которая определяет, какие операции можно выполнять с этим значением, как оно будет храниться и каким образом его можно обработать.

У Python есть стандартные типы данных, по которым определяются допустимые операции над ними и методы хранения для каждого из них.

Таблица 1. Стандартные типы данных в Python

Тип данных	Описание
bool	Логический тип данных, может принимать значения True или False
float	Вещественные числа, дробная часть отделяется точкой
int	Целые числа, их размер ограничен размером оперативной памяти
complex	Комплексные числа

str	Unicode-строки
list	Списки
tuple	Кортежи
dict	Словари
set	Множества

Идентификаторы, или переменные, в Python представляют собой имена, которые используются, чтобы обозначить разные объекты, такие как функции, классы и модули. Важно отметить, что для имен переменных нельзя использовать зарезервированные слова языка.

При формировании имени переменной допустимо использовать только английские буквы (A-Z, a-z), цифры (0-9) и символ подчеркивания (_). Однако первое место может занимать лишь буква или символ подчеркивания. Имя переменной может состоять из одной буквы, но не может составляться исключительно из символа подчеркивания, поскольку этот символ зарезервирован.

Ключевые слова в Python — это зарезервированные слова, которые имеют специальное значение и выполняют определенные функции в языке.

Зарезервированные символы позволяют управлять работой программы и манипулировать данными разных типов. Типы данных определяют, какие значения может принимать переменная и какие операции над ними можно выполнять, а ключевые слова помогают описывать структуру программы, условия, циклы и выполнять операции с переменными.

Эти слова нельзя использовать в качестве имен переменных, функций или других идентификаторов, так как они уже зарезервированы для выполнения конкретных задач.

Примеры зарезервированных слов можно найти в таблице, приведенной ниже.

Таблица 2. Зарезервированные слова

Ключевое слово	Значение
False	Ложь
class	Ключевое слово для определения класса
from	Предложение для импорта класса из модуля
with/as	Менеджер контекста

None	«Пустой» объект
and	Логическое И
break	Выход из цикла
continue	Переход на следующую итерацию цикла
def	Определение функции
del	Удаление объекта.
if	Если
not	Логическое НЕ
or	Логическое ИЛИ
return	Вернуть результат

В Python тип переменной автоматически меняется, если ей присваивается новое значение. Чтобы определить тип переменной, используется функция type().

Подходы к проверке типов в Python

1. **Статическая проверка типов** — тип переменной проверяется во время компиляции.
2. **Динамическая проверка типов** — тип переменной определяется, когда выполняется программа.

Также в Python есть модуль typing. Он дает возможность проверять типы переменных и поддерживать их аннотации.

В Python есть несколько встроенных функций, чтобы преобразовывать типы данных. Рассмотрим функцию int(s,[base]) Она преобразует s в целое число. Необязательный параметр base, приведенный в квадратных скобках, указывает на основание системы счисления (от 2 до 36). Результат работы зависит от типа вводимых данных.

В Python нет встроенного способа, чтобы определить константу. Значения переменных изменяются в любое время, и нет механизма, который позволяет указать, что переменная должна оставаться неизменной после инициализации. Тем не менее, существует общепринятая практика, чтобы создать константы в Python, основанная на соглашениях по наименованию.

PEP 8 (Python Enhancement Proposal 8) — это документ, который устанавливает стандарты оформления кода на Python.

Согласно рекомендации PEP 8, имена констант пишут большими буквами и используют подчеркивания для разделения слов.

Например, PI = 3.14159 или GRAVITY = 9.8.

Благодаря соглашению об именовании, другие программисты понимают, что переменная используется как константа и они не должны менять ее значение.

Выражения. Правила построения выражений. Работа с числовыми данными

Инструкция в Python представляет собой логическую команду, которую может интерпретировать и выполнить интерпретатор Python. Она может проявляться в виде выражения или оператора присваивания. Присваивание считается основополагающим компонентом языка и позволяет создавать и хранить объекты, основываясь на выражениях.

Выражение — это один из видов инструкции, который состоит из логической последовательности чисел, строк, объектов и операторов Python.

Как значения, так и переменные считаются выражениями. С помощью выражений можно выполнять различные операции, такие как сложение, вычитание и конкатенацию. Они также могут включать вызовы функций, которые возвращают определенные результаты.

Эта структура позволяет разработчикам гибко манипулировать данными и эффективно реализовывать логику программирования в своих приложениях.

Таблица 3. Значение арифметических операторов

Арифметический оператор	Значение
+	Сложение
-	Вычитание
*	Умножение
/	Деление с остатком
//	Целочисленное деление
%	Остаток от деления
**	Возведение в степень
=	Присвоение

Например,

A = 10 # присвоить A значение 10

B = 16 # присвоить B значение 16

print(A*B) # вывод 160

print(A+B) # вывод 26

print(A-B) # вывод -6

print(A/B) # вывод 0.625

print(A//B) # вывод 0

print(A%B) # вывод 10

print(A**B) # вывод 10000000000000000

Вопрос 4. Работа со строками. Работа с датами и временем. Функции преобразования данных

Строковый тип str в Python используют для работы с любыми текстовыми данными. Python автоматически определяет тип str по кавычкам — одинарным или двойным: stroka = 'Python'. Для решения многих задач строковую переменную нужно объявить заранее, пока не исполняется основная часть программы. Создать пустую переменную str просто: stroka = ' '. Кроме двойных и одинарных кавычек ', в Python используются и тройные ''' — в них заключают текст, который состоит из нескольких строк, или программный код.

Сложение и умножение строк

Строки можно складывать — эта операция называется **конкатенация**. Если надо, строку можно умножить на целое число — это **репликация**.

Пример 1. Конкатенация и репликация

```
>>> str1 = 'Python'
>>> str2 = ' - '
>>> str3 = 'самый гибкий язык программирования'
>>> print(str1 + str2 + str3)
Python - самый гибкий язык программирования
```

```
>>> stroka = '*** '
>>> print(stroka * 5)
*** *** *** *** ***
```

Каждую строку можно разделить на подстроки. Чтобы определить, входит ли какая-то подстрока в строку, используют оператор in.

Пример 2. Разделение на подстроки

```
>>> stroka = 'abrakadabra'
>>> print('abra' in stroka)
True
>>> print('zebra' in stroka)
False
```

Чтобы обратиться к определенному символу строки, используют индекс — порядковый номер элемента. Python поддерживает два типа индексации — положительную, при которой отсчет элементов начинается с 0 и с начала строки, и отрицательную, при которой отсчет начинается с -1 и с конца. Чтобы получить определенный элемент строки, нужно указать его индекс в квадратных скобках.

Пример 3. Элемент строки

```
>>> stroka = 'программирование'
>>> print(stroka[7])
м
>>> print(stroka[-1])
е
```

Индексы позволяют работать с отдельными элементами строк. Для работы с подстроками используют срезы, в которых задается нужный диапазон. Диапазон среза [a:b] начинается с первого указанного элемента a включительно, и заканчивается на последнем, только не включает b в результат.

Пример 4. Диапазон среза

```
>>> stroka = 'программирование'
>>> print(stroka[7:10])
мир
```

```
>>> stroka = 'программа'
>>> print(stroka[3:8])
грамм
```

Если не указать первый элемент диапазона [:b], срез выполнится с начала строки до позиции второго элемента b. Если второй элемент [a:] отсутствует, срез произойдет с позиции первого символа и до конца строки. Если не указать ни стартовую, ни финальную позиции среза, он будет равен исходной строке. Срез

[::-1] может оказаться очень полезным при решении задач, связанных с палиндромами.

Строки в Python относятся к неизменяемым типам данных. По этой причине попытка заменить символ по индексу ничего не даст.

Таблица 4. Методы работы со строками

Метод	Значение
join()	Разбить строку
split()	Преобразовать строку в список
partition()	Преобразовать строку в кортеж

Модули для работы с датами и временем в Python

1. Модуль datetime предоставляет классы для работы с датами и временем, такие как datetime, date, time и timedelta.
2. Модуль time предоставляет функции для работы со временем. С их помощью можно получить текущее время, настроить ожидание нужного временного отрезка и преобразовать представления времени.
3. Модуль calendar предоставляет функции для работы с календарями. С его помощью можно получить количество дней в месяце, определить день недели для заданной даты и форматировать даты разными способами.

Примеры использования модулей смотрите в таблице ниже.

Таблица 5. Использование модулей

Модуль	Пример использования
datetime.date	print(datetime.date(2012,12, 14))
	import datetime d = datetime.date(2012, 12, 14) print(d.year) print(d.day) print(d.month)
	import datetime print(datetime.date.today())
datetime.datetime	import datetime a = datetime.datetime(2017, 3, 5) print(a) b = datetime.datetime(2017, 3, 5, 12, 30, 10) print(b) d = datetime.datetime(2017, 3, 5, 12, 30, 10) print(d.year) print(d.second)

	print(d.hour)
	import datetime a = datetime.datetime.today() print(a) b = datetime.datetime.now() print(b)
	import datetime a = datetime.datetime.today().strftime("%Y%m%d") print(a) today = datetime.datetime.today() print(today.strftime("%m/%d/%Y")) print(today.strftime("%Y-%m-%d-%H.%M.%S")
datetime.timedelta	import datetime now = datetime.datetime.now() then = datetime.datetime(2017, 2, 26) delta = now - then print(delta.days) print(delta.seconds)
	seconds = delta.total_seconds() hours = seconds // 3600 print(hours) minutes = (seconds % 3600) // 60 print(minutes)
time	import time print(time.gmtime(0))
	time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=\ 0, tm_wday=3, tm_yday=1, tm_isdst=0)
time.ctime	import time print(time.ctime()) print(time.ctime(1384112639))

time.sleep	import time
	for x in range(5):
	  time.sleep(2)
	  print("Slept for 2 seconds")

time.strftime	import time a = time.strftime("%Y-%m-%d-%H.%M.%S", time.localtime()) print(a)
time.time	import time x = time.time() print(x)
	import time a = time.ctime(time.time()) print(a)

Преобразование типов данных в Python

Если мы складываем два числа, то это обычная операция сложения. Если же речь идет о двух строках, мы выполняем операцию их объединения. Однако, когда нужно сложить строку и число, Python не знает, как это сделать. В таком случае, если наша цель — объединить строку и число, надо преобразовать число в строку с помощью функции str().

Таблица 6. Преобразование типов данных в Python

Функции	Характеристика
ord()	Преобразует символы в целое число
hex()	Преобразует целое число в шестнадцатеричную строку
oct()	Преобразует целое число в восьмеричную строку
tuple()	Преобразует в кортеж
set()	Возвращает тип после преобразования в set
list()	Преобразует любой тип данных в тип списка
int()	Конвертирует данные в целочисленный формат
str()	Преобразовывает данные в строку
float()	Обрабатывает данные, которые представляют в виде строк с плавающей точкой или целых чисел