Самостоятельная работа. Модуль 8.

Милованов Дмитрий Михалович

«Разработка и отладка программного обеспечения». Тема 8.2. Инструменты отладки и тестирования.

|  |  |  |
| --- | --- | --- |
| №  п/п | Наименование  задания | Ответ |
| 1 | Изучение | 1. Левашов Пётр. Python с нуля. – СПб.: Питер, 2024. – 448 с. 2. Donald Norris. Python for Microcontrollers. 2017. – 284 c. 3. Алгоритмы и структуры данных на Python. – СПб.: Питер. -368 с. 4. Балабанов П.В. Программирование беспилотного летательного аппарата мультироторного типа. 2023. – 86 с. |
|  | дополнительных |
|  | материалов по |
|  | отладке и |
|  | тестированию. |
| 2 | Выполнить | 2.1. Опишите основные цели отладки и тестирования при разработке программного обеспечения для БЛА. |
|  | дополнительное | 1. исправление ошибок и дефектов в коде; 2. обеспечение соответствия требованиям заказчика; 3. выявление дефектов и ошибок; 4. повышение стабильности и надёжности работы программы; 5. оптимизация производительности и эффективности программного обеспечения; 6. улучшение качества кода и предотвращение будущих ошибок; |
|  | упражнение по |  |
|  | отладке и | 2.2. Опишите основные инструменты юнит-тестирования и интеграционного тестирования. |
|  | тестированию. | 1) Pytest(для Python) : позволяет писать компактные тесты.  2) JUnit(для Java): предоставляет аннотации.3)  NUnit(для C / C++) : хорошая поддержка интеграции.  4) MSTest(для .Net) : создание развёрнутых тестов.  5) CMocka(для C/C++) : тест низкоуровневого кода.  6) GTest(дляC/C++): выполнение тестов Google Test.  Интеграционное тестирование  1) Postman (тестирование API и сервисов).  2) Selenium (тестирование веб-приложений).  3) JUnit& Mockito(для тестирования на Java).  4) S pytest& pytest-django(для Python).  5) Jest (длятестирования кода на JavaScript).  6) Robot Framework (для тестирования API).  7) Cucumber (для описания работы модулей). |
|  |  | 2.3. В выбранной среде разработки (IDE) подготовьте небольшой проект на примере анализа данных с сенсоров БЛА (например, акселерометр, гироскоп, барометр или GPS). Приложите скринкопию кода и файл с кодом. |
|  |  | import time |
|  |  | def convert\_to\_decimal(degree\_str, direction):  """Преобразует строку с градусами в десятичный формат.""" # Преобразуем строку в float  degree\_value = float(degree\_str)  degrees = int(degree\_value // 100) |

|  |  |  |
| --- | --- | --- |
|  |  | minutes = degree\_value % 100 decimal = degrees + (minutes / 60.0) if direction in ['S', 'W']:  decimal \*= -1 return decimal  def data\_gps(gps\_data): """Обработка данных с GPS.  Args:  gps\_data: Строка данных в формате NMEA.  Returns:  Словарь с обработанными данными:   * latitude: Широта (градусы). * longitude: Долгота (градусы). * altitude: Высота (метры). * speed: Скорость (м/с). * timestamp: Время (секунды). """   try:  # Разделение строки NMEA по запятым data\_parts = gps\_data.split(",")  # Проверка на формат GGA  if data\_parts[0] == "$GPGGA": # Обработка данных GGA timestamp\_str = data\_parts[1] # Время фиксируется latitude = convert\_to\_decimal(data\_parts[2], data\_parts[3]) longitude = convert\_to\_decimal(data\_parts[4], data\_parts[5]) altitude = float(data\_parts[9])  speed = None # Скорость не доступна в GGA  # Форматирование даты для mktime  # Текущая дата, чтобы использовать её вместе со временем current\_time = time.localtime()  current\_date = time.strftime("%Y,%m,%d", current\_time).split(',')  year, month, day = int(current\_date[0]), int(current\_date[1]), int(current\_date[2]) |

|  |  |  |
| --- | --- | --- |
|  |  | # Преобразование времени  timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}", "%H%M%S.%f,%Y,%m,%d") timestamp\_seconds = time.mktime(timestamp)  # Проверка на формат RMC для извлечения скорости elif data\_parts[0] == "$GPRMC":  timestamp\_str = data\_parts[1]  latitude = convert\_to\_decimal(data\_parts[3], data\_parts[4]) longitude = convert\_to\_decimal(data\_parts[5], data\_parts[6])  speed = float(data\_parts[7]) \* 0.514444 # Преобразование узлов в м/с altitude = None # Высота недоступна в RMC  # Форматирование даты для mktime current\_time = time.localtime()  current\_date = time.strftime("%Y,%m,%d", current\_time).split(',')  year, month, day = int(current\_date[0]), int(current\_date[1]), int(current\_date[2])  # Преобразование времени  timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}", "%H%M%S.%f,%Y,%m,%d") timestamp\_seconds = time.mktime(timestamp)  else:  raise ValueError("Неизвестный формат NMEA")  # Возвращение обработанных данных return {  "latitude": latitude, "longitude": longitude, "altitude": altitude, "speed": speed,  "timestamp": timestamp\_seconds  }  except Exception as e:  print("Ошибка при обработке данных GPS:", e) return None |

|  |  |  |
| --- | --- | --- |
|  |  | # Реализация алгоритма  gps\_data\_gga = "$GPGGA,123519.487,3754.587,N,14507.036,W,1,08,0.9,545.4,M,46.9,M,,\*47" gps\_data\_rmc = "$GPRMC,123519.487,A,3754.587,N,14507.036,W,000.0,360.0,120419,,,D"  data\_gps\_gga = data\_gps(gps\_data\_gga) data\_gps\_rmc = data\_gps(gps\_data\_rmc)  for data\_source, data\_gps\_obj in zip(["GGA", "RMC"], [data\_gps\_gga, data\_gps\_rmc]): if data\_gps\_obj:  print(f"Обработанные данные из {data\_source}:") print(f"Широта: {data\_gps\_obj['latitude']}") print(f"Долгота: {data\_gps\_obj['longitude']}") print(f"Высота: {data\_gps\_obj['altitude']}") print(f"Скорость: {data\_gps\_obj['speed']}") print(f"Время: {data\_gps\_obj['timestamp']}")  else:  print(f"Некорректные данные GPS из {data\_source}.")  Обработанные данные: |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 3 | Разработка | В выбранной интегрированной среде разработки (IDE) подготовьте небольшой проект на примере анализа данных с сенсоров БЛА (например, акселерометр, гироскоп, барометр или GPS). Приложите скринкопию проекта и архив проекта.  Разработка проекта была проведена с применением юнит-тестирования, интеграционного тестирования и системного тестирования.   1. **Разработка проекта с применением юнит-тестирования.**    * был создан файл gps\_module.py    * был создан файл test\_gps.py    * в терминале интеграционной среды разработки Visual Studio Code была выполнена команда python –m unittest test\_gps.py для запуска теста.   Файл gps\_module.py def data\_gps(gps\_data):  """Обработка данных с GPS.  Args:  gps\_data: Строка данных в формате NMEA.  Returns: |
|  | небольшого |
|  | проекта с |
|  | использованием |
|  | инструментов |
|  | отладки и |
|  | тестирования. |

|  |  |  |
| --- | --- | --- |
|  |  | Словарь с обработанными данными:   * latitude: Широта (градусы). * longitude: Долгота (градусы). * altitude: Высота (метры). * speed: Скорость (м/с). * timestamp: Время (секунды). """   try:  # Разделение строки NMEA по запятым data\_parts = gps\_data.split(",")  # Проверка на формат GGA  if data\_parts[0] == "$GPGGA":  timestamp\_str = data\_parts[1] # Время фиксируется latitude = convert\_to\_decimal(data\_parts[2], data\_parts[3]) longitude = convert\_to\_decimal(data\_parts[4], data\_parts[5]) altitude = float(data\_parts[9])  speed = None # Скорость не доступна в GGA  # Форматирование даты для mktime current\_time = time.localtime()  year, month, day = current\_time.tm\_year, current\_time.tm\_mon, current\_time.tm\_mday timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}",  "%H%M%S.%f,%Y,%m,%d")  timestamp\_seconds = time.mktime(timestamp)  # Проверка на формат RMC для извлечения скорости |

|  |  |  |
| --- | --- | --- |
|  |  | elif data\_parts[0] == "$GPRMC": timestamp\_str = data\_parts[1]  latitude = convert\_to\_decimal(data\_parts[3], data\_parts[4]) longitude = convert\_to\_decimal(data\_parts[5], data\_parts[6])  speed = float(data\_parts[7]) \* 0.514444 # Преобразуем узлы в м/с altitude = None # Высота недоступна в RMC  # Форматирование даты для mktime current\_time = time.localtime()  year, month, day = current\_time.tm\_year, current\_time.tm\_mon, current\_time.tm\_mday timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}",  "%H%M%S.%f,%Y,%m,%d")  timestamp\_seconds = time.mktime(timestamp)  else:  raise ValueError("Неизвестный формат NMEA")  # Возвращение обработанных данных return {  "latitude": latitude, "longitude": longitude, "altitude": altitude, "speed": speed,  "timestamp": timestamp\_seconds  }  except Exception as e: |

|  |  |  |
| --- | --- | --- |
|  |  | print("Ошибка при обработке данных GPS:", e) return None  Файл test\_gps.py  import unittest  from gps\_module import data\_gps  class TestGPSFunctions(unittest.TestCase): def test\_gga\_data(self):  gps\_data\_gga =  "$GPGGA,123519.487,3754.587,N,14507.036,W,1,08,0.9,545.4,M,46.9,M,,\*47"  expected\_output = {  "latitude": 37.90978333333333,  "longitude": -145.11726666666667,  "altitude": 545.4, "speed": None,  "timestamp": 1731058519.0 # Значение времени может изменяться в зависимости от текущей даты  }  # Получение данных  output = data\_gps(gps\_data\_gga)  # Тестирование  self.assertAlmostEqual(output["latitude"], expected\_output["latitude"], places=7) self.assertAlmostEqual(output["longitude"], expected\_output["longitude"], places=7)  self.assertEqual(output["altitude"], expected\_output["altitude"]) |

|  |  |  |
| --- | --- | --- |
|  |  | self.assertIsNone(output["speed"])  def test\_rmc\_data(self): gps\_data\_rmc =  "$GPRMC,123519.487,A,3754.587,N,14507.036,W,000.0,360.0,120419,,,D"  expected\_output = {  "latitude": 37.90978333333333,  "longitude": -145.11726666666667,  "altitude": None, "speed": 0.0,  "timestamp": 1731058519.0 # Значение времени может изменяться в зависимости от текущей даты  }  # Получение данных  output = data\_gps(gps\_data\_rmc)  # Тестирование  self.assertAlmostEqual(output["latitude"], expected\_output["latitude"], places=7) self.assertAlmostEqual(output["longitude"], expected\_output["longitude"], places=7) self.assertIsNone(output["altitude"])  self.assertEqual(output["speed"], expected\_output["speed"])  if name == " main ": unittest.main()  Отчёт о выполненном задании представлен далее при ответе на вопрос №4.  **2) Разработка проекта с применением интеграционного тестирования.**  - был создан файл gps\_module.py |

|  |  |  |
| --- | --- | --- |
|  |  | * был создан файл integration\_test\_gps.py * в терминале интеграционной среды разработки Visual Studio Code была выполнена команда python –m unittest integration\_test\_gps.py для запуска теста.   Файл gps\_module.py import time  def convert\_to\_decimal(degree\_str, direction):  """Преобразование строки с градусами в десятичный формат.""" # Преобразование строки в float  degree\_value = float(degree\_str) degrees = int(degree\_value // 100) minutes = degree\_value % 100 decimal = degrees + (minutes / 60.0)  if direction in ['S', 'W']: decimal \*= -1  return decimal  def data\_gps(gps\_data): """Обработка данных с GPS.  Args:  gps\_data: Строка данных в формате NMEA.  Returns:  Словарь с обработанными данными:   * + latitude: Широта (градусы).   + longitude: Долгота (градусы).   + altitude: Высота (метры).   + speed: Скорость (м/с).   + timestamp: Время (секунды). |

|  |  |  |
| --- | --- | --- |
|  |  | """  try:  # Разделение строки NMEA по запятым data\_parts = gps\_data.split(",")  # Проверка на формат GGA  if data\_parts[0] == "$GPGGA":  timestamp\_str = data\_parts[1] # Время фиксируется latitude = convert\_to\_decimal(data\_parts[2], data\_parts[3]) longitude = convert\_to\_decimal(data\_parts[4], data\_parts[5]) altitude = float(data\_parts[9])  speed = None # Скорость не доступна в GGA  # Форматирование даты для mktime current\_time = time.localtime()  year, month, day = current\_time.tm\_year, current\_time.tm\_mon, current\_time.tm\_mday timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}",  "%H%M%S.%f,%Y,%m,%d")  timestamp\_seconds = time.mktime(timestamp)  # Проверка на формат RMC для извлечения скорости elif data\_parts[0] == "$GPRMC":  timestamp\_str = data\_parts[1]  latitude = convert\_to\_decimal(data\_parts[3], data\_parts[4]) longitude = convert\_to\_decimal(data\_parts[5], data\_parts[6])  speed = float(data\_parts[7]) \* 0.514444 # Преобразование узлов в м/с altitude = None # Высота недоступна в RMC  # Форматирование даты для mktime current\_time = time.localtime()  year, month, day = current\_time.tm\_year, current\_time.tm\_mon, current\_time.tm\_mday timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}", |

|  |  |  |
| --- | --- | --- |
|  |  | "%H%M%S.%f,%Y,%m,%d")  timestamp\_seconds = time.mktime(timestamp)  else:  raise ValueError("Неизвестный формат NMEA")  # Возвращение обработанных данных return {  "latitude": latitude, "longitude": longitude, "altitude": altitude, "speed": speed,  "timestamp": timestamp\_seconds  }  except Exception as e:  print("Ошибка при обработке данных GPS:", e) return None  Файл integration\_test\_gps.py import unittest  from gps\_module import data\_gps  class TestGPSIntegration(unittest.TestCase): def test\_gga\_data\_integration(self):  gps\_data\_gga = "$GPGGA,123519.487,3754.587,N,14507.036,W,1,08,0.9,545.4,M,46.9,M,,\*47"  expected\_output = {  "latitude": 37.90978333333333,  "longitude": -145.11726666666667, |

|  |  |  |
| --- | --- | --- |
|  |  | "altitude": 545.4, "speed": None,  # Значение времени не проверяется в тесте  }  output = data\_gps(gps\_data\_gga)  # Выполнение тестирования  self.assertAlmostEqual(output["latitude"], expected\_output["latitude"], places=7) self.assertAlmostEqual(output["longitude"], expected\_output["longitude"], places=7) self.assertEqual(output["altitude"], expected\_output["altitude"]) self.assertIsNone(output["speed"])  def test\_rmc\_data\_integration(self): gps\_data\_rmc =  "$GPRMC,123519.487,A,3754.587,N,14507.036,W,000.0,360.0,120419,,,D"  expected\_output = {  "latitude": 37.90978333333333,  "longitude": -145.11726666666667,  "altitude": None, "speed": 0.0,  # Значение времени не проверяется в тесте  }  output = data\_gps(gps\_data\_rmc)  # Выполнение тестирования  self.assertAlmostEqual(output["latitude"], expected\_output["latitude"], places=7) self.assertAlmostEqual(output["longitude"], expected\_output["longitude"], places=7) self.assertIsNone(output["altitude"])  self.assertEqual(output["speed"], expected\_output["speed"])  if name == " main ": unittest.main() |

|  |  |  |
| --- | --- | --- |
|  |  | **3) Разработка проекта с применением системного тестирования.**   * был создан файл gps\_module.py * был создан файл system\_test\_gps.py * в терминале интеграционной среды разработки Visual Studio Code была выполнена команда python –m unittest system\_test\_gps.py для запуска теста.   Файл gps\_module.py import time  def convert\_to\_decimal(degree\_str, direction):  """Преобразование строки с градусами в десятичный формат.""" if not degree\_str: # Проверка на пустую строку  raise ValueError("Строка с градусами пуста")  # Преобразование строки в float degree\_value = float(degree\_str) degrees = int(degree\_value // 100) minutes = degree\_value % 100 decimal = degrees + (minutes / 60.0)  if direction in ['S', 'W']: decimal \*= -1  return decimal  def data\_gps(gps\_data): """Обработка данных с GPS. |

|  |  |  |
| --- | --- | --- |
|  |  | Args:  gps\_data: Строка данных в формате NMEA.  Returns:  Словарь с обработанными данными:   * latitude: Широта (градусы). * longitude: Долгота (градусы). * altitude: Высота (метры). * speed: Скорость (м/с). * timestamp: Время (секунды). """   try:  # Разделение строки NMEA по запятым data\_parts = gps\_data.split(",")  # Проверка на формат GGA  if data\_parts[0] == "$GPGGA":  # Проверка минимальной длины данных и проверка на пустые поля  if len(data\_parts) < 10 or not data\_parts[1] or not data\_parts[2] or not data\_parts[4] or not data\_parts[9]:  raise ValueError("Недостаточно данных для анализа GGA.")  timestamp\_str = data\_parts[1] # Время фиксируется latitude = convert\_to\_decimal(data\_parts[2], data\_parts[3]) longitude = convert\_to\_decimal(data\_parts[4], data\_parts[5]) altitude = float(data\_parts[9])  speed = None # Скорость недоступна в GGA  # Форматирование даты для mktime current\_time = time.localtime()  year, month, day = current\_time.tm\_year, current\_time.tm\_mon, current\_time.tm\_mday timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}", |

|  |  |  |
| --- | --- | --- |
|  |  | "%H%M%S.%f,%Y,%m,%d")  timestamp\_seconds = time.mktime(timestamp)  # Проверка на формат RMC для извлечения скорости elif data\_parts[0] == "$GPRMC":  # Проверка минимальной длины данных и проверка на пустые поля  if len(data\_parts) < 10 or not data\_parts[1] or not data\_parts[3] or not data\_parts[5]: raise ValueError("Недостаточно данных для анализа RMC.")  timestamp\_str = data\_parts[1]  latitude = convert\_to\_decimal(data\_parts[3], data\_parts[4]) longitude = convert\_to\_decimal(data\_parts[5], data\_parts[6])  speed = float(data\_parts[7]) \* 0.514444 if data\_parts[7] else 0.0 # Преобразуем узлы в  м/с  altitude = None # Высота недоступна в RMC  # Форматирование даты для mktime current\_time = time.localtime()  year, month, day = current\_time.tm\_year, current\_time.tm\_mon, current\_time.tm\_mday timestamp = time.strptime(f"{timestamp\_str},{year},{month},{day}",  "%H%M%S.%f,%Y,%m,%d")  timestamp\_seconds = time.mktime(timestamp)  else:  raise ValueError("Неизвестный формат NMEA")  # Возвращение обработанных данных return {  "latitude": latitude, "longitude": longitude, "altitude": altitude,  "speed": speed, |

|  |  |  |
| --- | --- | --- |
|  |  | "timestamp": timestamp\_seconds  }  except Exception as e:  print("Ошибка при обработке данных GPS:", e) return None  Файл system\_test\_gps.py import unittest  from gps\_module import data\_gps  class TestGPSSystem(unittest.TestCase): def test\_valid\_gga\_data(self):  gps\_data\_gga = "$GPGGA,123456.78,4916.45,N,12311.12,W,1,12,0.5,30.0,M,0.0,M,,\*47"  expected\_output = {  "latitude": 49 + (16.45 / 60), # 49.27425  "longitude": -(123 + (11.12 / 60)), # -123.18533333333334  "altitude": 30.0,  "speed": None,  }  output = data\_gps(gps\_data\_gga)  # Выполнение тестирования  self.assertAlmostEqual(output["latitude"], expected\_output["latitude"], places=7) self.assertAlmostEqual(output["longitude"], expected\_output["longitude"], places=7) self.assertEqual(output["altitude"], expected\_output["altitude"]) self.assertIsNone(output["speed"])  def test\_valid\_rmc\_data(self): gps\_data\_rmc = |

|  |  |  |
| --- | --- | --- |
|  |  | "$GPRMC,123519.487,A,3754.587,N,14507.036,W,000.0,360.0,120419,,,D"  expected\_output = {  "latitude": 37 + (54.587 / 60), # 37.90978333333333  "longitude": -(145 + (7.036 / 60)), # -145.11726666666667  "altitude": None, "speed": 0.0,  }  output = data\_gps(gps\_data\_rmc)  # Выполнение тестирования  self.assertAlmostEqual(output["latitude"], expected\_output["latitude"], places=7) self.assertAlmostEqual(output["longitude"], expected\_output["longitude"], places=7) self.assertIsNone(output["altitude"])  self.assertEqual(output["speed"], expected\_output["speed"])  def test\_invalid\_data(self):  gps\_data\_invalid = "$GPGGA,12345,,N,,W,1,08,0.9,,,M,46.9,M,,47"  output = data\_gps(gps\_data\_invalid)  self.assertIsNone(output) # Ожидается, что выход будет None  def test\_edge\_case(self): gps\_data\_edge\_case =  "$GPRMC,000000.000,V,0000.0000,N,00000.0000,W,000.0,000.0,010101,,,D"  output = data\_gps(gps\_data\_edge\_case)  self.assertIsNotNone(output) # Ожидается, что выход не будет None  self.assertEqual(output["latitude"], 0.0) # Ожидается, что широта будет 0 self.assertEqual(output["longitude"], 0.0) # Ожидается, что долгота будет 0  if name == " main ": unittest.main() |

|  |  |  |
| --- | --- | --- |
| 4 | Подготовка отчёта о выполненных заданиях. | По итогам проведённого тестирования с применением юнит-теста, интеграционного теста и системного теста можно сделать несколько основных выводов относительно качества, надежности и функциональности разработанного небольшого проекта.   1. Успешно был выполнен юнит-тест. Результат теста показал, что функция реализована правильно. Далее представлен результат теста.     Целью юнит-теста является обеспечение корректного функционирования каждой отдельной единицы кода.   1. Успешно был выполнен интеграционный тест. Результат теста показал, что функция, которая вызывается в ходе системы, возвращает ожидаемые результаты наборов данных. Далее представлен результат теста.      1. Успешно был выполнен системный тест. Результат теста показал, как подготовленный алгоритм справляется с некорректными строками и крайними случаями (например, пустыми строками, недостаточным количеством полей или недостаточным количеством данных для анализа). |

|  |  |  |
| --- | --- | --- |
|  |  | Целью системного теста является проверка работы системы в целом с различными условиями, включая проверку корректного и некорректного ввода, можно добавлять дополнительные тестовые случаи, чтобы  охватить больше сценариев. |