```
!pip install radon
```

```
Collecting radon
  Downloading radon-6.0.1-py2.py3-none-any.whl.metadata (8.2 kB)
Collecting mando<0.8,>=0.6 (from radon)
  Downloading mando-0.7.1-py2.py3-none-any.whl.metadata (7.4 kB)
Collecting colorama>=0.4.1 (from radon)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from mando<0.8,>=0.6->radon) (1.16.0)
Downloading radon-6.0.1-py2.py3-none-any.whl (52 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.8/52.8 kB 2.1 MB/s eta 0:00:00
Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Downloading mando-0.7.1-py2.py3-none-any.whl (28 kB)
Installing collected packages: mando, colorama, radon
Successfully installed colorama-0.4.6 mando-0.7.1 radon-6.0.1
```

```python
from radon.metrics import mi_visit, mi_rank
from google.colab import files
```

```python
def save_text_as_python_file(text, file_name):
    with open(file_name, 'w') as file:
        file.write(text)
```

```python
 beforeChange = """
 import json
 import smtplib
 import unittest
 from unittest.mock import patch, MagicMock
 from email.mime.text import MIMEText
 from email.mime.multipart import MIMEMultipart
 import os
 import uuid

 class User:
     def _init_(self, email, username):
         self.email = email
         self.username = username

     def to_dict(self):
         return {"email": self.email, "username": self.username}

     @staticmethod
     def from_dict(data):
         return User(data['email'], data['username'])

 class Event:
     def _init_(self, event_name, event_id=None, filepath='events.json'):
         self.event_name = event_name
         self.event_id = event_id if event_id else str(uuid.uuid4())
         self.filepath = filepath
         self.subscribers = self.load_from_json()

     def add_subscriber(self, user):
         self.subscribers.append(user)
         self.save_to_json()

     def save_to_json(self):
         all_events = self.load_all_events()
         all_events[self.event_id] = {
             "event_name": self.event_name,
             "subscribers": [user.to_dict() for user in self.subscribers]
         }
         with open(self.filepath, 'w') as f:
             json.dump(all_events, f, indent=4)

     def load_from_json(self):
         all_events = self.load_all_events()
         event_data = all_events.get(self.event_id, {})
         return [User.from_dict(data) for data in event_data.get("subscribers", [])]

     def load_all_events(self):
         if not os.path.exists(self.filepath):
             return {}
         with open(self.filepath, 'r') as f:
             return json.load(f)
```

```

class Publisher:
    def send_email_to_all(self, event, subject, message, smtp_server, smtp_port, smtp_user, smtp_pass):
        for user in event.subscribers:
            self.send_email(user.email, subject, message, smtp_server, smtp_port, smtp_user, smtp_pass)

    @staticmethod
    def send_email(to_email, subject, message, smtp_server, smtp_port, smtp_user, smtp_pass):
        msg = MIMEMultipart()
        msg['From'] = smtp_user
        msg['To'] = to_email
        msg['Subject'] = subject

        msg.attach(MIMEText(message, 'plain'))

        try:
            server = smtplib.SMTP(smtp_server, smtp_port)
            server.starttls()
            server.login(smtp_user, smtp_pass)
            text = msg.as_string()
            server.sendmail(smtp_user, to_email, text)
            server.quit()
            print(f"Email sent to {to_email}")
        except Exception as e:
            print(f"Failed to send email to {to_email}: {e}")

# Unit Tests
class TestUser(unittest.TestCase):
    def test_to_dict(self):
        user = User("user1@example.com", "user1")
        self.assertEqual(user.to_dict(), {"email": "user1@example.com", "username": "user1"})

    def test_from_dict(self):
        data = {"email": "user2@example.com", "username": "user2"}
        user = User.from_dict(data)
        self.assertEqual(user.email, "user2@example.com")
        self.assertEqual(user.username, "user2")

class TestEvent(unittest.TestCase):
    def setUp(self):
        self.filepath = 'test_events.json'
        self.event1 = Event("Event 1", filepath=self.filepath)
        self.event2 = Event("Event 2", filepath=self.filepath)
        self.user1 = User("user1@example.com", "user1")
        self.user2 = User("user2@example.com", "user2")

    def tearDown(self):
        try:
            os.remove(self.filepath)
        except OSError:
            pass

    def test_add_subscriber(self):
        self.event1.add_subscriber(self.user1)
        self.assertIn(self.user1, self.event1.subscribers)

    def test_save_and_load_json(self):
        self.event1.add_subscriber(self.user1)
        self.event1.add_subscriber(self.user2)
        self.event1.save_to_json()

        loaded_event = Event("Event 1", event_id=self.event1.event_id, filepath=self.filepath)
        self.assertEqual(len(loaded_event.subscribers), 2)
        self.assertEqual(loaded_event.subscribers[0].email, "user1@example.com")
        self.assertEqual(loaded_event.subscribers[1].email, "user2@example.com")

    def test_subscribe_to_specific_event(self):
        self.event1.add_subscriber(self.user1)
        self.event2.add_subscriber(self.user2)
        self.event1.save_to_json()
        self.event2.save_to_json()

        loaded_event1 = Event("Event 1", event_id=self.event1.event_id, filepath=self.filepath)
        loaded_event2 = Event("Event 2", event_id=self.event2.event_id, filepath=self.filepath)

        self.assertEqual(len(loaded_event1.subscribers), 1)
        self.assertEqual(len(loaded_event2.subscribers), 1)
        self.assertEqual(loaded_event1.subscribers[0].email, "user1@example.com")
        self.assertEqual(loaded_event1.subscribers[0].email, "user1@example.com")
        self.assertEqual(loaded_event2.subscribers[0].email, "user2@example.com")

class TestPublisher(unittest.TestCase):
    @patch('smtplib.SMTP')
    def test_send_email_to_all(self, mock_smtp):
        event = Event("Sample Event", filepath='test_event_subscribers.json')
        user1 = User("user1@example.com", "user1")
        user2 = User("user2@example.com", "user2")
        event.add_subscriber(user1)
        event.add_subscriber(user2)

        publisher = Publisher()
        smtp_server = "smtp.example.com"
        smtp_port = 587
        smtp_user = "your_email@example.com"
        smtp_pass = "your_password"

        publisher.send_email_to_all(
            event,
            "Event Notification",
            "This is a notification for an upcoming event.",
            smtp_server,
            smtp_port,
            smtp_user,
            smtp_pass
        )

        self.assertEqual(mock_smtp.call_count, 1)
        instance = mock_smtp.return_value
        self.assertEqual(instance.sendmail.call_count, 2)

if _name_ == "_main_":
    unittest.main()
"""

afterChange = '''
from pymongo import MongoClient
import uuid

class User:
    def _init_(self, email, username):
        self.email = email
        self.username = username

    def to_dict(self, event_id=None):
        user_dict = {"email": self.email, "username": self.username}
        if event_id:
            user_dict["event_id"] = event_id
        return user_dict

    @staticmethod
    def from_dict(data):
        return User(data['email'], data['username'])

class Event:
    def _init_(self, event_name, event_id=None, db=None, collection_name=None):
        self.event_name = event_name
        self.event_id = event_id if event_id else str(uuid.uuid4())
        self.db = db
        self.collection = db[f"{collection_name}"]
        self.subscribers = self.load_from_db()

    def add_subscribers(self, users):
        for user in users:
            if user not in self.subscribers:
                self.subscribers.append(user)
        self.save_to_db()

    def save_to_db(self):
        event_data = {
            "event_id": self.event_id,
            "event_name": self.event_name,
            "subscribers": [user.to_dict() for user in self.subscribers]
        }
        self.collection.replace_one({"event_id": self.event_id}, event_data, upsert=True)

    def load_from_db(self):
        event_data = self.collection.find_one({"event_id": self.event_id})
        if event_data:
            return [User.from_dict(data) for data in event_data.get("subscribers", [])]
        return []

# Connect to MongoDB
client = MongoClient("mongodb+srv://thierryarnold41:qknFM59lZ4UmJ1fk@cluster0.fnhwx8z.mongodb.net/?retryWrites=true&w=majority&appName=C
db = client['eventDB']

# Create an Event and add multiple subscribers
event = Event("Sample Event", db=db, collection_name="eventRead")
users = [
    User("user1@example.com", "user1"),
    User("user2@example.com", "user2"),
    User("user3@example.com", "user3")
]
event.add_subscribers(users)

event = Event("Sample Event", db=db, collection_name="eventWrite")
users = [
    User("user1@example.com", "user1"),
    User("user2@example.com", "user2"),
    User("user3@example.com", "user3")
]
event.add_subscribers(users)

# Retrieve and print the subscribers
event_from_db = Event("Sample Event", db=db)
subscribers = event_from_db.load_from_db()
for subscriber in subscribers:
    # Print subscriber details along with the event ID
    print(subscriber.to_dict(event_id=event.event_id))
'''
```

```
file_name = 'beforeChange.py'
file_name2 = 'afterChange.py'
save_text_as_python_file(beforeChange, file_name)
save_text_as_python_file(afterChange, file_name2)
```

```
def calculate_maintainability_index(file_path):
    with open(file_path, 'r') as file:
        code = file.read()

    # Calculate the maintainability index
    maintainability_index = mi_visit(code, False)
    rank = mi_rank(maintainability_index)

    print(f"File: {file_path}")
    print(f"Maintainability Index: {maintainability_index}")
    print(f"Rank: {rank}")
```

```
# Calculate and print the maintainability index for the saved file
calculate_maintainability_index(file_name)
print("--------------------------------")
calculate_maintainability_index(file_name2)
```

```
File: beforeChange.py
Maintainability Index: 48.22626975433067
Rank: A
--------------------------------
File: afterChange.py
Maintainability Index: 71.18261571921715
Rank: A
```