

Code Translation using Generative AI

Estimated time needed: 30 minutes

As a software developer, you may be very comfortable coding in a particular language. But often you have requests from client, to use a different language than what you are comfortable with. In the past, in such situations, you would either have had to learn the language or have to give up on the client and lose business. But today, generative AI can make it much simpler for developers.

Please remember that the prompts that you feed to generative AI are like a conversation with a subject matter expert and the consecutive prompts depend on the previous ones and the response received. Change the wording if required to get a specific desired result. The example showcases one possible chat conversation to attain the objective.

Learning Objectives

At the end of the lab, you will be able to:

1. Use generative AI to translate the code written in one language to another, retaining the logic and workflow as is.
2. Convert JavaScript CRUD code to Python CRUD code without any scripting in Python.

Prerequisites

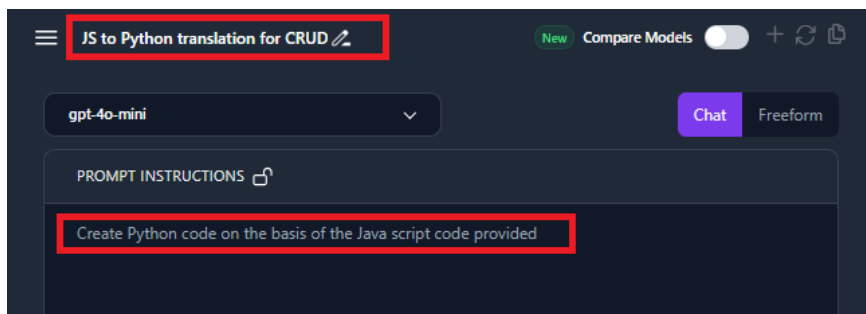
Know at least one programming language. The lab was created with some sample code in JavaScript. It will be good if you know how to read the JavaScript code.

Set up chat title and prompt instruction

When the chat session opens, a new chat conversation begins. Give the chat conversation an appropriate title. This will help you revisit the chat conversation. It is a good practice to segregate the conversations topically as it will help in continuing the conversation at a later point.

Also, provide prompt instructions that are specific to the conversation in this particular lab. Let's get started with the task of translating JavaScript code to Python.

Please note that generative AI is an evolving field. As you attempt the labs, your experience and output might be different than what is seen here.



Tool capability to translate

You are already comfortable coding in JavaScript. You have created the code in JavaScript. You will now use gen AI model to create the same in Python.

Firstly, you need to determine if the gen AI tool has the capability to do the required translation. In the prompt type:

I have a requirement to code in Python. If I provide the JavaScript code, can you help me create the Python equivalent of the code?

If the GenAI model has the ability to translate, you will get an affirmative response in line with what is shown below. Python is one of the most commonly used languages, and this task of translating from JS to Python will be doable. But if there are other newer or scarcely used languages, you will need to use your judgment to perceive confidence in the response and decide whether to continue to use the tool, either entirely or as a support tool. Iterative prompts can be used to ensure the correctness of the responses.

▼ [Click here to view the sample response generated](#)

Absolutely! Please provide the JavaScript code you'd like to convert to Python, and I'll help you create the equivalent code.

Provide the code to Gen AI

Paste the following code into the prompt and send it. This process will take a few seconds as it needs to process the code given and then give the equivalent.

```
// Employee class
class Employee {
  constructor(id, name, age, position) {
    this.id = id;
    this.name = name;
    this.age = age;
    this.position = position;
  }
}
// Employee data storage
let employees = [];
// Create employee
function createEmployee(id, name, age, position) {
  const employee = new Employee(id, name, age, position);
  employees.push(employee);
}
```

```

}
// Read employee by ID
function readEmployee(id) {
  return employees.find((employee) => employee.id === id);
}
// Update employee by ID
function updateEmployee(id, updatedData) {
  const employee = employees.find((employee) => employee.id === id);
  if (employee) {
    Object.assign(employee, updatedData);
  }
}

// Delete employee by ID
function deleteEmployee(id) {
  employees = employees.filter((employee) => employee.id !== id);
}

```

It will produce an output as Python code similar to what is shown below.

▼ Click here to view the sample response generated

Here's the Python equivalent of the provided JavaScript code for managing an Employee class and related functions:

```

Employee class
class Employee:
    def __init__(self, id, name, age, position):
        self.id = id
        self.name = name
        self.age = age
        self.position = position

Employee data storage
employees = []

Create employee
def create_employee(id, name, age, position):
    employee = Employee(id, name, age, position)
    employees.append(employee)

Read employee by ID
def read_employee(id):
    return next((employee for employee in employees if employee.id == id), None)

Update employee by ID
def update_employee(id, updated_data):
    employee = next((employee for employee in employees if employee.id == id), None)
    if employee:
        for key, value in updated_data.items():
            setattr(employee, key, value)

Delete employee by ID
def delete_employee(id):
    global employees
    employees = [employee for employee in employees if employee.id != id]

Example usage
if __name__ == "__main__":
    create_employee(1, "Alice", 30, "Developer")
    create_employee(2, "Bob", 25, "Designer")
    print(read_employee(1)) # Output: <__main__.Employee object at ...>

    update_employee(1, {"age": 31, "position": "Senior Developer"})

    print(read_employee(1).age) # Output: 31

    delete_employee(2)
    print(read_employee(2)) # Output: None

```

Additional Details:

- **Data Storage:** In this implementation, the employees list is used to store instances of the Employee class.
- **Functions:** Each function corresponds to the operations defined in the JavaScript version: creating, reading, updating, and deleting employees.
- **Example Usage:** The if __name__ == "__main__": block demonstrates how to use the functions with example data.
- **Global Keyword:** The global keyword is used in delete_employee to modify the employees list defined outside the function scope.

Feel free to modify the example usage part to test different scenarios as needed!

This is with the assumption that the JavaScript code has been typed properly. If there are minor coding mistakes, gen AI has the ability to correct itself and give the right output.

There are often a few lines of explanation given at the end of the response. This will not be the same every time, as the responses are AI-generated and not programmed. The essence of the message is, however, consistent. If there is no explanation, you can prompt for an explanation of the code rendered.

Explain the python code

You can skip this step if the explanation offered at the end of the previous step suffices. If required, prompt for an explanation of the code.

Explain each step in the python code produced to help me understand the code.

The output would be similar to what is shown below. You can iteratively prompt for more specific questions you may have.

▼ Click here to view the sample response generated

Sure! Let's break down the Python code step by step to understand how it works.

1. Employee Class Definition

```

class Employee:
    def __init__(self, id, name, age, position):

```

```

self.id = id
self.name = name
self.age = age
self.position = position

```

- **Class Definition:** We define a class named `Employee` that represents an employee.
- **Constructor (`__init__` method):** This is a special method that initializes a new instance of the `Employee` class. It takes parameters `id`, `name`, `age`, and `position`, which are assigned to instance variables (`self.id`, `self.name`, etc.).

2. Employee Data Storage

```
employees = []
```

- **List Initialization:** We create an empty list called `employees` to store instances of the `Employee` class. This will serve as our data storage.

3. Create Employee Function

```

def create_employee(id, name, age, position):
    employee = Employee(id, name, age, position)
    employees.append(employee)

```

- **Function Definition:** This function takes parameters for the employee's details.
- **Creating an Employee Instance:** We create a new `Employee` instance using the provided parameters.
- **Appending to List:** The new employee object is added to the `employees` list.

4. Read Employee Function

```

def read_employee(id):
    return next((employee for employee in employees if employee.id == id), None)

```

- **Function Definition:** This function takes an `id` as a parameter to find a specific employee.
- **Finding the Employee:** We use a generator expression to iterate over the `employees` list and find the first employee whose `id` matches the provided `id`.
- **Return Value:** If found, the employee object is returned; otherwise, `None` is returned if no match is found.

5. Update Employee Function

```

def update_employee(id, updated_data):
    employee = next((employee for employee in employees if employee.id == id), None)
    if employee:
        for key, value in updated_data.items():
            setattr(employee, key, value)

```

- **Function Definition:** This function takes an `id` and a dictionary `updated_data` containing the fields to update.
- **Finding the Employee:** Similar to the `read_employee` function, we find the employee by `id`.
- **Updating Fields:** If the employee is found, we iterate over the `updated_data` dictionary. The `setattr` function is used to dynamically update the employee's attributes based on the keys in the dictionary.

6. Delete Employee Function

```

def delete_employee(id):
    global employees
    employees = [employee for employee in employees if employee.id != id]

```

- **Function Definition:** This function takes an `id` to identify which employee to delete.
- **Global Keyword:** We declare `employees` as global to modify the list defined outside the function.
- **Filtering the List:** We create a new list that includes only those employees whose `id` does not match the provided `id`, effectively removing the specified employee.

7. Example Usage

```

if __name__ == "__main__":
    create_employee(1, "Alice", 30, "Developer")
    create_employee(2, "Bob", 25, "Designer")
    print(read_employee(1)) # Output: <__main__.Employee object at ...>

    update_employee(1, {"age": 31, "position": "Senior Developer"})

    print(read_employee(1).age) # Output: 31

    delete_employee(2)
    print(read_employee(2)) # Output: None

```

- **Main Check:** The `if __name__ == "__main__":` block ensures that this code runs only when the script is executed directly, not when imported as a module.
- **Creating Employees:** We create two employees, Alice and Bob, using the `create_employee` function.
- **Reading Employee:** We read Alice's information and print the employee object (it will display the memory address).
- **Updating Employee:** We update Alice's age and position using the `update_employee` function.
- **Printing Updated Information:** We print Alice's updated age to verify the change.
- **Deleting Employee:** We delete Bob using the `delete_employee` function and check if he still exists by attempting to read his information, which should return `None`.

Summary

This Python code replicates the functionality of the provided JavaScript code, allowing you to create, read, update, and delete employee records. Each function is designed to perform a specific operation, and the use of a list for storage is a straightforward approach to managing the employee data.

Versions of Python

It is important to know which version of Python the code will work with. There might be requirements to provide older versions for backward compatibility, especially when you are going to create software that fits into other existing software. Type the next prompt as:

Let me know the versions of Python that this code will run smoothly on.

The output would be similar to what is given below.

▼ Click here to view the sample response generated

The provided Python code should run smoothly on any version of Python 3, specifically from Python 3.6 onwards. Here's why:

1. **F-strings:** Although the code does not use f-strings, they were introduced in Python 3.6 and are often used for formatted string outputs. If you decide to enhance the code with formatted outputs in the future, you'll need at least Python 3.6.
2. **Type Hinting:** The code does not include type hints, which were introduced in Python 3.5. If you wish to add type hints for better readability and type checking, Python 3.5 and later would be suitable.
3. **General Features:** The core features used in the provided code, such as lists, classes, and functions, have been available since the early versions of Python 3. Therefore, any version from 3.0 onward should technically run the code, but Python 3.6 and later is recommended for better performance and additional features.

Recommended Versions:

- **Python 3.6:** Introduced f-strings and other performance improvements.
- **Python 3.7:** Added data classes, which can be a more concise way to define classes like Employee.
- **Python 3.8 and later:** Introduced the walrus operator (:=), positional-only parameters, and other enhancements.

Conclusion

For the best experience and access to the latest features, using Python 3.8 or later is advisable. However, the code will work just fine on any version starting from Python 3.6. Always ensure that you are using a version that is still supported for security and performance reasons.

Include version information in code

It is easier to include the Python version information in the file. Type the next prompt as:

Include the python versions in which the code will be supported, as a comment in the appropriate place.

It will reproduce the same code as before, with the comment statement about the version included.

▼ Click here to view the sample response generated

Certainly! Here's the updated Python code with comments included to specify the versions of Python that support the code:

```
# Employee class
class Employee:
    def __init__(self, id, name, age, position):
        self.id = id
        self.name = name
        self.age = age
        self.position = position

# Employee data storage
employees = []

# Create employee
def create_employee(id, name, age, position):
    employee = Employee(id, name, age, position)
    employees.append(employee)

# Read employee by ID
def read_employee(id):
    return next((employee for employee in employees if employee.id == id), None)

# Update employee by ID
def update_employee(id, updated_data):
    employee = next((employee for employee in employees if employee.id == id), None)
    if employee:
        for key, value in updated_data.items():
            setattr(employee, key, value)

# Delete employee by ID
def delete_employee(id):
    global employees
    employees = [employee for employee in employees if employee.id != id]

# Example usage
if __name__ == "__main__":
    create_employee(1, "Alice", 30, "Developer")
    create_employee(2, "Bob", 25, "Designer")
    print(read_employee(1)) # Output: <__main__.Employee object at ...>

    update_employee(1, {"age": 31, "position": "Senior Developer"})

    print(read_employee(1).age) # Output: 31

    delete_employee(2)
    print(read_employee(2)) # Output: None

# Supported Python Versions:
# - Python 3.6 and later: Basic functionality
# - Python 3.7 and later: Data classes (not used but relevant for future enhancements)
# - Python 3.8 and later: Newer features and performance improvements
```

Summary of Comments:

- The comments at the end of the code specify the supported Python versions:
 - **Python 3.6 and later:** This is the minimum version required for the code to run.
 - **Python 3.7 and later:** Mentioned for potential future enhancements, such as using data classes.
 - **Python 3.8 and later:** Noted for access to newer features and performance improvements.

This should help clarify the compatibility of the code with different Python versions!

Including comments

Comments are extremely important for code readability. It is always good coding practice to include comments explaining what the code does. Type the next prompt as:

Include comments in the code which will help in readability of the code

▼ Click here to view the sample response generated

Sure! Here's the updated Python code with additional comments throughout the code to enhance readability and understanding:

```
# Employee class definition
class Employee:
    def __init__(self, id, name, age, position):
        # Initialize employee attributes
        self.id = id          # Unique identifier for the employee
        self.name = name      # Employee's name
        self.age = age        # Employee's age
        self.position = position # Employee's job position

# Employee data storage
employees = [] # List to store employee instances
# Create a new employee and add to the employee list
def create_employee(id, name, age, position):
    # Create a new instance of Employee
    employee = Employee(id, name, age, position)
    # Append the new employee instance to the employees list
    employees.append(employee)

# Read an employee's information by their ID
def read_employee(id):
    # Find and return the employee with the matching ID, or None if not found
    return next((employee for employee in employees if employee.id == id), None)

# Update an existing employee's information by their ID
def update_employee(id, updated_data):
    # Find the employee by ID
    employee = next((employee for employee in employees if employee.id == id), None)
    # If employee is found, update their attributes
    if employee:
        for key, value in updated_data.items():
            setattr(employee, key, value) # Update the attribute dynamically

# Delete an employee from the list by their ID
def delete_employee(id):
    global employees # Declare global to modify the employees list
    # Filter out the employee with the matching ID
    employees = [employee for employee in employees if employee.id != id]

# Example usage of the employee management system
if __name__ == "__main__":
    # Create some employees
    create_employee(1, "Alice", 30, "Developer")
    create_employee(2, "Bob", 25, "Designer")
    # Read and print Alice's information
    print(read_employee(1)) # Output: <__main__.Employee object at ...>

    # Update Alice's age and position
    update_employee(1, {"age": 31, "position": "Senior Developer"})

    # Print Alice's updated age
    print(read_employee(1).age) # Output: 31

    # Delete Bob from the employee list
    delete_employee(2)
    # Attempt to read Bob's information, which should return None
    print(read_employee(2)) # Output: None

# Supported Python Versions:
# - Python 3.6 and later: Basic functionality
# - Python 3.7 and later: Data classes (not used but relevant for future enhancements)
# - Python 3.8 and later: Newer features and performance improvements
```

Summary of Added Comments:

- **Class and Function Comments:** Each class and function now has comments explaining its purpose and the key operations being performed.
- **Attribute Comments:** Comments in the Employee class explain the attributes being initialized.
- **Example Usage Comments:** The example usage section includes comments that describe what each operation is doing, making it easier to follow the flow of the code.

These comments will help anyone reading the code to understand its structure and functionality more easily.

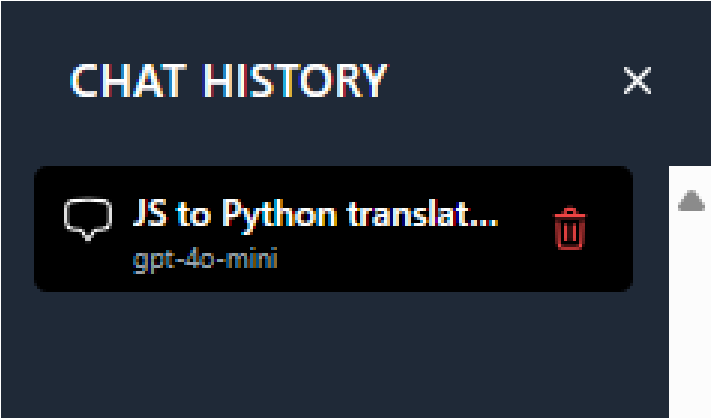
Conclusion

Congratulations! You have created Python code without knowing or knowing little about the language. If you are happy with the output produced, you may copy the code, use it, and engage generative AI to enhance the code with other functionalities. If you want to improvise, you may ask additional relevant questions. Generative AI depends on external sources to supplement the responses with more facts and realism. However, it is your prerogative as a software developer to run the software and see if it is created as per the requirements.

You can always visit the conversation during the live session using the history option by clicking on the burger menu on the top-left.



Then choose the chat by title among all the labs listed out.



Author(s)

Rama
Lavanya