1. Write a class called RestaurantCheck. It should have the following: (use OOPs concepts)

• Fields called check_number, sales_tax_percent, subtotal, table_number, and server_name representing an identification for the check, the bill without tax added, the sales tax percentage, the table number, and the name of the server. • A constructor that sets the values of all four fields • A method called calculate_total that takes no arguments (besides self) and returns the total bill including sales tax. • A method called print_check that writes to a file called check###.txt, where ### is the check number and writes information about the check to that file, formatted like below:

```
Check Number: 443
Sales tax: 6.0%
Subtotal: $23.14
Total: $24.53
Table Number: 17
Server: Sonic the Hedgehog
```

Test the class by creating a RestaurantCheck object and calling the print_check() method.

In [6]:
```
class RestaurantCheck:
    def __init__(self, check_number, subtotal, sales_tax_percent, table_number, server_n
        self.check_number = check_number
        self.subtotal = subtotal
        self.sales_tax_percent = sales_tax_percent
        self.table_number = table_number
        self.server_name = server_name

    def calculate_total(self):
        sales_tax = self.subtotal * (self.sales_tax_percent / 100)
        total = self.subtotal + sales_tax
        return total

    def print_check(self):
        filename = f"check{self.check_number}.txt"
        with open(filename, "w") as file:
            file.write(f"Check Number: {self.check_number}\n")
            file.write(f"Sales tax: {self.sales_tax_percent}%\n")
            file.write(f"Subtotal: ${self.subtotal:.2f}\n")
            file.write(f"Total: ${self.calculate_total():.2f}\n")
            file.write(f"Table Number: {self.table_number}\n")
            file.write(f"Server: {self.server_name}\n")


check = RestaurantCheck(443, 23.14, 6.0, 17, "Sonic the Hedgehog")
check.print_check()
print("Check file generated.")
```

```
Check file generated.
```

In [10]:
```
import re

def validate_input(phone_number, name, email, date):
    # Phone number validation (10 digits)
    phone_regex = r'^\d{10}$'
    e.match(phone_regex, phone_number):
```

```
        return False

    # Name validation (first character uppercase)
    name_regex = r'^[A-Z][a-zA-Z\s]+$'
    if not re.match(name_regex, name):
        return False

    # Email validation
    email_regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if not re.match(email_regex, email):
        return False

    # Date validation (DD-MM-YYYY)
    date_regex = r'^\d{2}-\d{2}-\d{4}$'
    if not re.match(date_regex, date):
        return False

    return True

# Get user input
phone_number = input("Enter the phone number: ")
name = input("Enter the name: ")
email = input("Enter the email: ")
date = input("Enter the date (DD-MM-YYYY): ")

# Validate input
if validate_input(phone_number, name, email, date):
    print("Inputs are valid.")
else:
    print("Inputs are invalid.")
```

```
Enter the phone number: 1234567890
Enter the name: Adhitya
Enter the email: asdfgh@gmail.com
Enter the date (DD-MM-YYYY): 12-08-2001
Inputs are valid.
```