

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, sales_tax_percent, subtotal, table_number, server_name):
    self.check_number = check_number
    self.sales_tax_percent = sales_tax_percent
    self.subtotal = subtotal
    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):
    total = self.calculate_total()
    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: ${total:.2f}\n")
        file.write(f"Table Number: {self.table_number}\n")
        file.write(f"Server: {self.server_name}\n")
    print(f"Check printed to {filename}")
```

```
In [8]: # Create a check instance
RestaurantCheck = RestaurantCheck(443, 6.0, 23.14, 17, "Sonic the Hedgehog")

# Calculate the total
total = RestaurantCheck.calculate_total()
print("Total:", total)

# Print the check
check.print_check()
```

```
Total: 24.5284
Check printed to check443.txt
```

2. Write a Regular Expression Python function to Validate

Phone No, (Must be 10 digits)
Name, (first Char must be uppercase)
E-Mail, (abc@abc.com)
Date .(DD-MM-YYYY)

```
In [18]: import re

def validate_input(phone_number, name, email, date):
    # Validate phone number (must be 10 digits)
    if not re.match(r'^\d{10}$', phone_number):
        return False

    # Validate name (first character must be uppercase)
    if not re.match(r'^[A-Z][a-zA-Z]*$', name):
        return False

    # Validate email address (abc@abc.com)
    if not re.match(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$', email):
        return False
```

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

return True
```

```
In [24]: phone_number = "1234567890"
name = "Pavan"
email = "pavan.reddy@gmail.com"
date = "06-09-2023"

if validate_input(phone_number, name, email, date):
    print("Inputs are valid.")
else:
    print("Inputs are not valid.")
```

Inputs are valid.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js