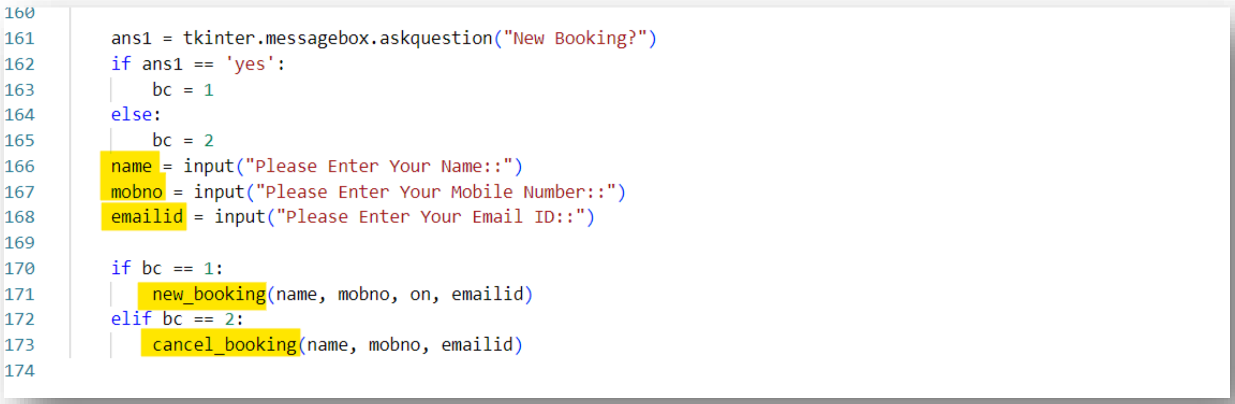
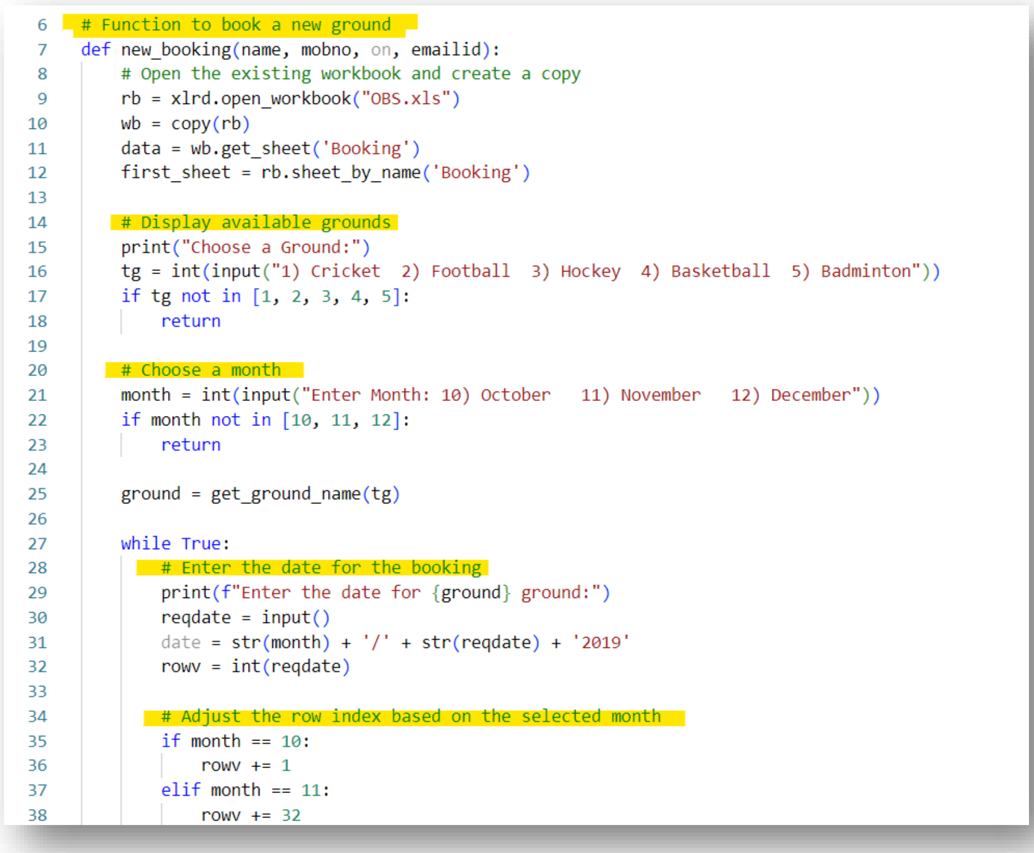
CCD - Clean Code Development

- The code is easy to read and understand because the variable and function names are self-explanatory. The functions purpose is communicated through meaningful names such as ‘new_booking’ and ‘cancel_booking’.



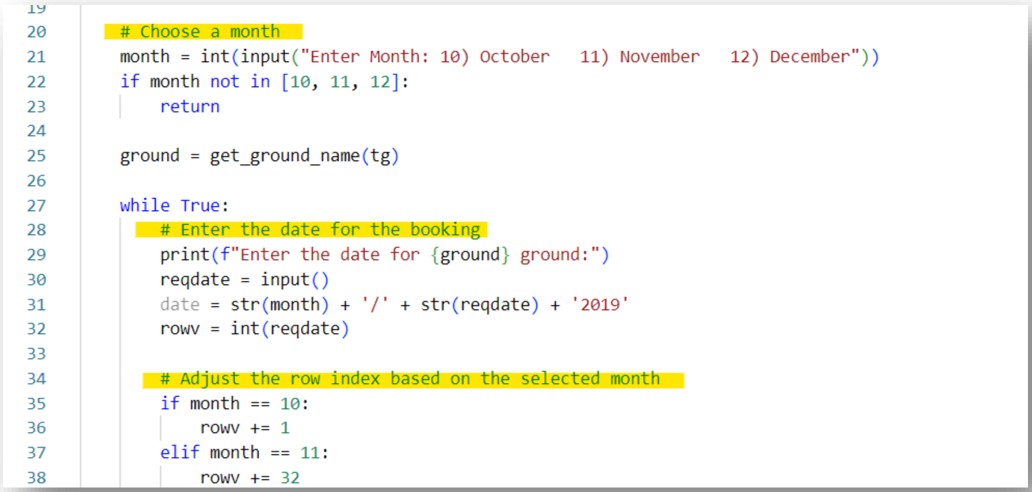
```python

    ans1 = tkinter.messagebox.askquestion("New Booking?")
    if ans1 == 'yes':
        bc = 1
    else:
        bc = 2
    name = input("Please Enter Your Name::")
    mobno = input("Please Enter Your Mobile Number::")
    emailid = input("Please Enter Your Email ID::")

    if bc == 1:
        new_booking(name, mobno, on, emailid)
    elif bc == 2:
        cancel_booking(name, mobno, emailid)

```

- I used comments(#), they provide explanations where necessary for understanding.



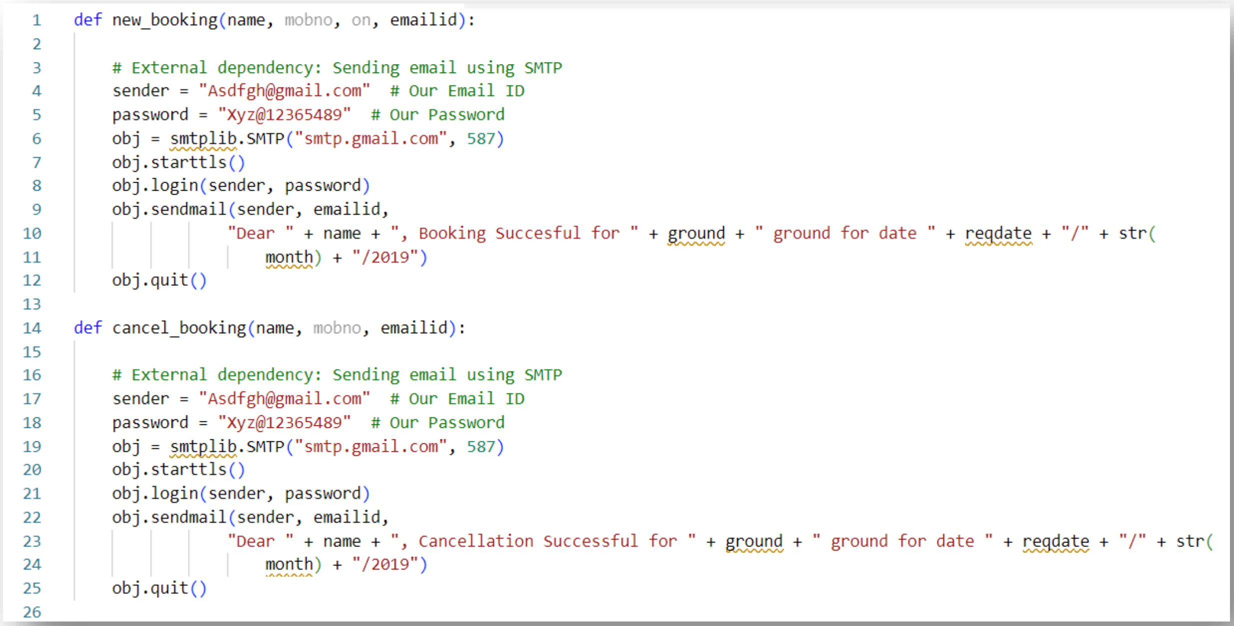
```python
# Function to book a new ground
def new_booking(name, mobno, on, emailid):
    # Open the existing workbook and create a copy
    rb = xlrd.open_workbook("OBS.xls")
    wb = copy(rb)
    data = wb.get_sheet('Booking')
    first_sheet = rb.sheet_by_name('Booking')

    # Display available grounds
    print("Choose a Ground:")
    tg = int(input("1) Cricket  2) Football  3) Hockey  4) Basketball  5) Badminton"))
    if tg not in [1, 2, 3, 4, 5]:
        return

    # Choose a month
    month = int(input("Enter Month: 10) October   11) November   12) December"))
    if month not in [10, 11, 12]:
        return

    ground = get_ground_name(tg)

    while True:
        # Enter the date for the booking
        print(f"Enter the date for {ground} ground:")
        reqdate = input()
        date = str(month) + '/' + str(reqdate) + '2019'
        rowv = int(reqdate)

        # Adjust the row index based on the selected month
        if month == 10:
            rowv += 1
        elif month == 11:
            rowv += 32
```

- Using Proper Indentation like “,” and spaces helps the user to easily understand the code and make it more robust and readable.



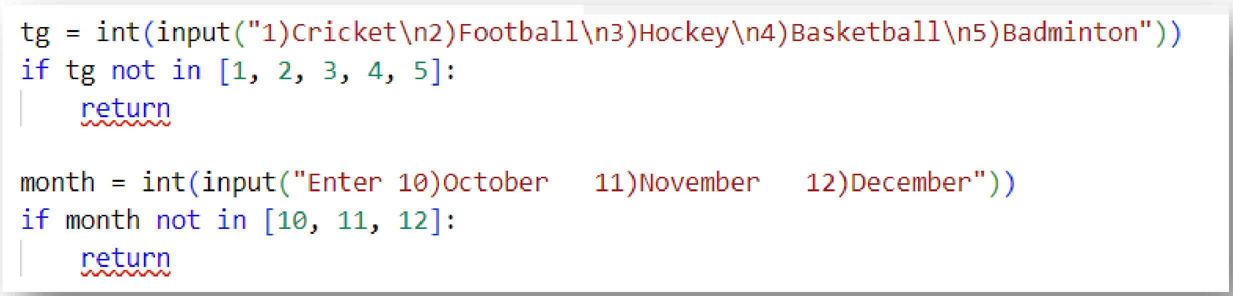
```python
    # Choose a month
    month = int(input("Enter Month: 10) October   11) November   12) December"))
    if month not in [10, 11, 12]:
        return

    ground = get_ground_name(tg)

    while True:
        # Enter the date for the booking
        print(f"Enter the date for {ground} ground:")
        reqdate = input()
        date = str(month) + '/' + str(reqdate) + '2019'
        rowv = int(reqdate)

        # Adjust the row index based on the selected month
        if month == 10:
            rowv += 1
        elif month == 11:
            rowv += 32
```

- External dependencies, such as email sending using SMTP, are encapsulated within functions, promoting separation of concerns.



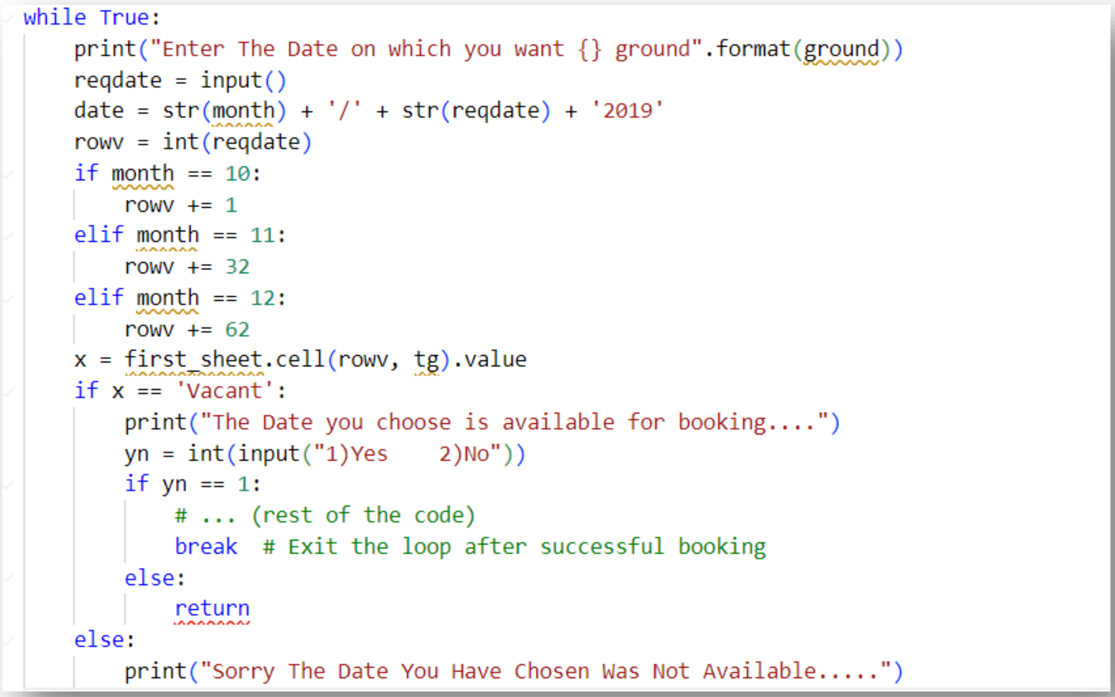
```python
def new_booking(name, mobno, on, emailid):

    # External dependency: Sending email using SMTP
    sender = "Asdfgh@gmail.com"  # Our Email ID
    password = "Xyz@12365489"  # Our Password
    obj = smtplib.SMTP("smtp.gmail.com", 587)
    obj.starttls()
    obj.login(sender, password)
    obj.sendmail(sender, emailid,
                "Dear " + name + ", Booking Succesful for " + ground + " ground for date " + reqdate + "/" + str(
                    month) + "/2019")
    obj.quit()

def cancel_booking(name, mobno, emailid):

    # External dependency: Sending email using SMTP
    sender = "Asdfgh@gmail.com"  # Our Email ID
    password = "Xyz@12365489"  # Our Password
    obj = smtplib.SMTP("smtp.gmail.com", 587)
    obj.starttls()
    obj.login(sender, password)
    obj.sendmail(sender, emailid,
                "Dear " + name + ", Cancellation Successful for " + ground + " ground for date " + reqdate + "/" + str(
                    month) + "/2019")
    obj.quit()

```

- The code includes error handling for scenarios where the user provides invalid inputs, preventing unexpected crashes. Error handling is implemented in the code when the user provides invalid inputs, such as choosing an invalid ground number or an invalid month. Here is an example:
  If the user provides an invalid input, the code takes appropriate actions (e.g., returning from the function), preventing unexpected crashes due to incorrect user inputs. This demonstrates a proactive approach to handling potential errors and ensuring a smoother user experience.



```
tg = int(input("1)Cricket\n2)Football\n3)Hockey\n4)Basketball\n5)Badminton"))
if tg not in [1, 2, 3, 4, 5]:
    return

month = int(input("Enter 10)October   11)November   12)December"))
if month not in [10, 11, 12]:
    return
```

- The use of “break” statements within loops helps exit the loop when necessary, improving control flow.
  The use of "break" statements is employed in the code to exit the loop when a specific condition is met. Here's an example, the "break" statement is used to exit the loop when a successful booking is made. This helps improve the control flow by terminating the loop once the desired condition is met, avoiding unnecessary iterations and enhancing the overall readability of the code.



```
while True:
    print("Enter The Date on which you want {} ground".format(ground))
    reqdate = input()
    date = str(month) + '/' + str(reqdate) + '2019'
    rowv = int(reqdate)
    if month == 10:
        rowv += 1
    elif month == 11:
        rowv += 32
    elif month == 12:
        rowv += 62
    x = first_sheet.cell(rowv, tg).value
    if x == 'Vacant':
        print("The Date you choose is available for booking....")
        yn = int(input("1)Yes    2)No"))
        if yn == 1:
            # ... (rest of the code)
            break  # Exit the loop after successful booking
        else:
            return
    else:
        print("Sorry The Date You Have Chosen Was Not Available.....")
```