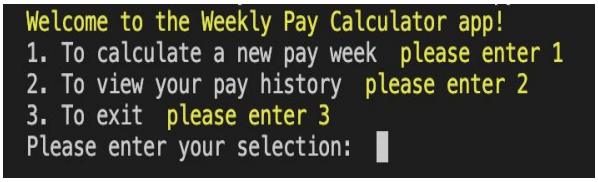
Terminal Application

Scott Taylor - 12141 - Coder Academy

Hello and welcome to the presentation of my Terminal Application.

Terminal Application

Main Menu



When the app is run in the terminal the first display that will appear for the user is the main menu that gives the user the option to make a selection of 1, 2 or 3. Looking at the code, the first option runs the pay calculator function, the second option will run the function to view the pay history in that been saved to the comma separated value (CSV) document. The third option with end the while loop and close the application.

```
# Displays a greeting for the user
print(f"{fg('yellow')}Welcome to the Weekly Pay Calculator app!{attr('reset')}")
# Creates a function that runs the menu for a users seection
def start_menu():
    print(f"1. To calculate a new pay week {fg('yellow')} please enter 1 {attr('reset')}")
    print(f"2. To view your pay history {fg('yellow')} please enter 2 {attr('reset')}")
    print(f"3. To exit {fg('yellow')} please enter 3 {attr('reset')}")
    selection = input("Please enter your selection:  ")
    return selection
# Runs a loop to get the users selection, if the user enters 1 it starts the calculator,
# 2 it accesses the csv file, 3 it ends the application
user_selection = ""
while user_selection != "3":
    user_selection = start_menu()
    if (user_selection == "1"):
        pay_calculator(file_name)

    elif (user_selection == "2"):
        view_pay_history(file_name)

    elif (user_selection == "3"):
        print(f"You are now {fg('yellow')}exiting.{attr('reset')}")
        continue
    else:
        print(f"{fg('red')} Please enter a number 1-3.{attr('reset')}")
        continue
    input(f"{fg('yellow')}Press enter {attr('reset')}to continue.....")

print(f"{fg('yellow')}Thank you for using the pay calculator!{attr('reset')}")
```

We will start by looking at the main menu that is the first thing that will be seen by the user when they run the application. Here the user can make a choice of 3 options, calculate a new pay week, view their pay history or exit the application. To the right of the screen we have the code for this menu, as you can see its a while loop being run while the user selection is not 3. Anything outside of 1, 2 or 3 asks the user to input again. So now lets look at what happens if the user enters 1 as their selection. It will run the pay calculator function.

Terminal Application

Pay Calculator

```
file_name = "pay_history.csv"
# When the user selection is 1 the pay calculator is called
def pay_calculator(file_name):
    # starts the weekly pay at zero, then calls the pay week function to get the starting date of the pay week,
    # then calls the user base rate function to get the user to input the rate of pay for the week.
    weekly_accumulated = 0
    pay_day = ""
    pay_week(file_name)
    base_rate = user_base_rate()
    # Runs a while loop getting each day worked and ending if the user inputs that theyre finished.
    while pay_day != "finished" or "Finished":
        # Depending on which day the user enters it will call that day of the weeks function and then add the returned value to the weekly accumulation.
        pay_day = input(f"Please enter a {fg('yellow')}day{attr('reset')}you've worked, or {fg('yellow')}'finished'{attr('reset')}when your pay weeks complete: ")

        # Monday
        if pay_day in ["monday", "Monday", "mon", "Mon"]:
            monday_pay = monday(file_name)
            weekly_accumulated += monday_pay
            print(f"{fg('yellow')}Weekly pay{attr('reset')} so far: $",format(weekly_accumulated,".2f"))

        # Tuesday
        elif pay_day in ["tuesday", "Tuesday", "tue", "Tue", "tues", "Tues"]:
            tuesday_pay = tuesday(file_name)
            weekly_accumulated += tuesday_pay
            print(f"{fg('yellow')}Weekly pay{attr('reset')} so far: $",format(weekly_accumulated,".2f"))

        # Wednesday
        elif pay_day in ["wednesday", "Wednesday", "wed", "Wed", "weds", "Weds"]:
            wednesday_pay = wednesday(file_name)
            weekly_accumulated += wednesday_pay
            print(f"{fg('yellow')}Weekly pay{attr('reset')} so far: $",format(weekly_accumulated,".2f"))
```

```
Please enter starting date of pay week in the format, DD-MM-YYYY : 17-04-2023
Please enter your base rate of pay : 36.52
Please enter a day you've worked, or 'finished' when your pay weeks complete: Monday
```

When the user chooses option 1 and the pay calculator function is called. The code on the left is what's inside of this function. It will initialise the weekly accumulated pay variable at zero, set the payday variable as an empty string for the loop and then call the functions for the pay week to get the date that the pay week starts at and then run the base rate function to get the users base rate of pay, once these variables are all set the loop begins, getting a day of the week input from the user that will then run that days function. The above display is what the user will see in the app.

So as we can see here this is the code for the pay calculator function, it will set the accumulated weekly pay at zero, the pay day as an empty string for the loop of the users input and run the functions for the starting date of the pay week and the users base rate of pay. Once its done this the user will then input the first day they worked on that week as you can see from the display at the top of the screen.

Terminal Application
Pay Calculator cont.

```
# Pay week function to determine the starting date of the pay week, with try except to ensure correct date input, then writes to csv.
def pay_week(file_name):
    date_inputs = ''
    while True:
        date_inputs = input(f"Please enter starting date of pay week in the format, {fg('yellow')}DD-MM-YYYY{attr('reset')} : ")
        try:
            date_outputs = datetime.strptime(date_inputs, "%d-%m-%Y").date()
            with open(file_name, "a")as pay_file:
                writer = csv.writer(pay_file)
                writer.writerow(["Pay week: ", date_outputs])
            break
        except ValueError:
            print(f"{fg('red')}That is not a valid date.{attr('reset')}")
```

Displayed here are the 2 functions that run at the start of the pay calculator, to obtain from the user the starting date of the pay week and base rate of pay. Both of these functions run a loop that will continue to ask the user for their input until its the type of input we need with try and except blocks, once the user enters a number or float, its then returned to the pay calculator function, where the date enter by the user, once correct, will write that date to the start of the csv file for the users pay history.

```
# Gets user input of thier base rate, runs try/except to make sure the value is a float and not negative.
def user_base_rate():
    global base_rate
    base_rate = ''
    while True:
        input_rate = (input(f"Please enter your {fg('yellow')}base rate{attr('reset')} of pay : "))
        try:
            base_rate = float(input_rate)
            if base_rate <= 0:
                raise ValueError
            break
        except ValueError:
            print(f"{fg('red')} '{input_rate}' is not a valid base rate of pay, please re-enter{attr('reset')}")
    return base_rate;
```

The 2 functions called at the start of the pay calculator are these 2 blocks of code, the top is the pay week function looping through inputs to make sure the correct date format is entered, running a try and except statement to keep the application from crashing from errors, writing the correct date and breaking the loop once receiving proper input. The bottom block obtains the users base rate of pay with a try and except block making sure to get a number, potentially a float from the user and making sure its not a negative one to keep the app from crashing from an error.

Terminal Application

Pay Calculator cont.

```
# Creates a function to calculate how much pay the user gets on any given day thats entered based on:
# which shift they worked, then writes it to the csv
# MONDAY
def monday(file_name):
    user_shift_worked("Monday")
    user_hours_worked("Monday")
    ph_penalties("Monday")

    if public_holiday in ["y", "Y", "yes", "Yes"]:
        monday_pay = public_holiday_pay(base_rate, hours_worked)
        print(f"{fg('yellow')}Monday's{attr('reset')} pay is: $",format(monday_pay,".2f"))
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Monday", format(monday_pay,".2f")])
            return monday_pay

    elif public_holiday in ["n", "N", "no", "No"] and shift_worked in ["day", "Day"]:
        monday_pay = day_shift_pay(base_rate, hours_worked)
        print(f"{fg('yellow')}Monday's{attr('reset')} pay is: $",format(monday_pay,".2f"))
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Monday", format(monday_pay,".2f")])
            return monday_pay

    elif public_holiday in ["n", "N", "no", "No"] and shift_worked in ["afternoon", "Afternoon"]:
        monday_pay = arvo_shift_pay(base_rate, hours_worked)
        print(f"{fg('yellow')}Monday's{attr('reset')} pay is: $",format(monday_pay,".2f"))
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Monday", format(monday_pay,".2f")])
            return monday_pay

    elif public_holiday in ["n", "N", "no", "No"] and shift_worked in ["night", "Night"]:
        monday_pay = night_shift_pay(base_rate, hours_worked)
        print(f"{fg('yellow')}Monday's{attr('reset')} pay is: $",format(monday_pay,".2f"))
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Monday", format(monday_pay,".2f")])
            return monday_pay
```

When the user enters a day they've worked into the pay calculator, it will call that day of the weeks function, the code for that function is on the left. This function will ask the user which shift they worked, how many hours they worked on that shift and if that shift was a public holiday. Depending on the shift and public holiday it will meet one of the 4 criteria in the if/else block, each of these blocks try to cover as many user entries that are considered the same, once it does it will calculate the potential shift loading and/or public holiday loading based on the hours worked for that day and then display that days pay and the accumulation for the week to the user in the app as well as write that daily value to the csv document for the users pay history.

```
What shift did you work on Monday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Monday : 7.6
Was Monday a public holiday?  y/n.  Y
Monday's pay is: $ 693.88
Weekly pay so far: $ 693.88
```

Once the user enters the day of the week they worked it will call that days function. We can see the functions code on the left side of this slide, the days function will call 3 more functions at the start that gets the users inputs for shift worked, hours worked and whether or not it was a public holiday. All of the these inputs allow the days function to activate the matching block of code in its if/else statement. Depending on which block it runs it will have a calculation function that will calculate and return that days pay back to the main calculator function

Terminal Application

Pay Calculator cont.

```
# Gets user input to see if it was a public holiday, runs if/else to make sure the input is a valid string.
def ph_penalties(day_of_week):
    global public_holiday
    public_holiday = ''
    while True:
        input_ph = input(f"Was {day_of_week} a {fg('yellow')}public holiday?{attr('reset')}  y/n.  ")
        if input_ph in ["y", "Y", "yes", "Yes", "n", "N", "no", "No"]:
            public_holiday = input_ph
            break
        else:
            print(f"{fg('red')} '{input_ph}' is not a valid responce, please enter y or n{attr('reset')}")
    return public_holiday;
```

```
# Gets user input of thier hours worked, runs try/except to make sure the value is a float and less than minimum hours.
def user_hours_worked(day_of_week):
    global hours_worked
    hours_worked = ''
    while True:
        input_hours = (input(f"Please enter {fg('yellow')}hours worked {attr('reset')}on {day_of_week} : "))
        try:
            hours_worked = float(input_hours)
            if hours_worked < 4:
                raise ValueError
            break
        except ValueError:
            print(f"{fg('red')} '{input_hours}' is not a valid amount of hours worked, please re-enter{attr('reset')}")
    return hours_worked;
```

```
# Gets user input of the shift they worked, runs if/else to make sure the input is a valid string.
def user_shift_worked(day_of_week):
    global shift_worked
    shift_worked = ''
    while True:
        input_shift = input(f"What {fg('yellow')}shift{attr('reset')} did you work on {day_of_week}? Please enter 'Day', 'Afternoon' or 'Night'.  ")
        if input_shift in ["Day", "day", "afternoon", "Afternoon", "night", "Night"]:
            shift_worked = input_shift
            break
        else:
            print(f"{fg('red')} '{input_shift}' is not a valid shift worked, please re-enter{attr('reset')}")
    return shift_worked;
```

The functions called inside each of the day of the week functions are these 3 functions. The first loops until it get a yes/no answer that matches the criteria and breaks and returns it once it does. The next get the users hours worked for the day, it loops until it get a value thats a number and once thats 4 or greater as thats the minimum amount of hours that can be worked, then returns it to the daily function. The last loops until the user inputs a shift that matches - day, afternoon or night. Once it receives that value it breaks and returns it to the daily function.

This is the code for the 3 functions called at the start of the days pay function. All of these loop through the input until it gets the correct input that string matches or runs through a try/except statement and then returns the input to the daily pay function.

Terminal Application

Pay Calculator cont.

```
# Calculates Public Holiday loading rate of pay
def public_holiday_pay(base_rate, hours_worked):
    ph_pay = (base_rate * 2.5) * hours_worked
    return ph_pay
# Calculates Day shift rate of pay
def day_shift_pay(base_rate, hours_worked):
    ds_pay = base_rate  * hours_worked
    return ds_pay
# Calculates Afternoon shift loading rate of pay
def arvo_shift_pay(base_rate, hours_worked):
    as_pay = (base_rate * 1.15)  * hours_worked
    return as_pay
# Calculates Night shift loading rate of pay
def night_shift_pay(base_rate, hours_worked):
    ns_pay = (base_rate * 1.25)  * hours_worked
    return ns_pay
# Calculates Saturday loading rate of pay
def sat_loading_pay(base_rate, hours_worked):
    satl_pay = (base_rate * 1.5) * hours_worked
    return satl_pay
# Calculates Sunday loading rate of pay
def sun_loading_pay(base_rate, hours_worked):
    sunl_pay = (base_rate * 1.8) * hours_worked
    return sunl_pay
```

To get the pay to return for each day there are separate calculations for each of the day functions based on which inputs the user has will run a different calculation that's inside of that days if/else. Those calculations are on the left. The only days that don't run one of those calculations are the night shift pays for saturday and sunday that have 2 hours of pay for the start of that day and then the rest of their shift based on how many hours they worked that night.

```
elif public_holiday in ["n", "N", "no" "No"] and shift_worked in ["night", "Night"]:
    saturday_pay = ((base_rate * 1.5) * 2) + ((base_rate * 1.8) * (hours_worked - 2))
    print(f"{fg('yellow')}Saturday's{attr('reset')} pay is: $",format(saturday_pay,".2f"))
    with open(file_name, "a")as pay_file:
        writer = csv.writer(pay_file)
        writer.writerow(["Saturday", format(saturday_pay,".2f")])
        return saturday_pay
```

```
elif public_holiday in ["n", "N", "no" "No"] and shift_worked in ["night", "Night"]:
    sunday_pay = ((base_rate * 1.8) * 2) + ((base_rate * 1.25) * (hours_worked - 2))
    print(f"{fg('yellow')}Sunday's{attr('reset')} pay is: $",format(sunday_pay,".2f"))
    with open(file_name, "a")as pay_file:
        writer = csv.writer(pay_file)
        writer.writerow(["Sunday", format(sunday_pay,".2f")])
        return sunday_pay
```

The calculations code for the days pay function are on the left of this slide, its basically just calculating the loadings of shift, weekend or public holidays. The only calculations that run in the actual days pay code blocks are the night shift for the weekend as the loadings cross over on saturday and sunday shifts, those blocks are displayed here.

Terminal Application

Pay Calculator cont.

```
# Full tax calculations based on users tax bracket once weekly pay is determined, to return gross and net pay to the user.
def weekly_pay(weekly_accumulated):
    yearly_income = (weekly_accumulated / 7) * 365
    medilevy = yearly_income * .02
    weekly_medilevy = (medilevy / 365) * 7

    if yearly_income <= 18200:
        print(f"Your {fg('yellow')}weekly pay{attr('reset')} is: $",format(weekly_accumulated,".2f"))
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Weekly Total Gross: ", format(weekly_accumulated,".2f")])


    elif yearly_income > 18200 and yearly_income <= 29032:
        yearly_tax = (yearly_income - 18200) * .19
        weekly_tax = (yearly_tax / 365) * 7
        weekly_tax = float(format(weekly_tax,".0f"))
        net_pay = weekly_accumulated - weekly_tax
        net_pay = format(net_pay,".2f")
        print(f"Your {fg('yellow')}gross weekly pay{attr('reset')} is: $",format(weekly_accumulated,".2f"))
        print(f"Your {fg('yellow')}net weekly pay{attr('reset')} is: ${net_pay} (Tax: {weekly_tax})")
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Weekly Total Gross: ", format(weekly_accumulated,".2f")])
            writer.writerow(["Weekly Total Net: ", net_pay])
```

Once the user has input that they've finished their pay week the weekly pay function will be called. This function will take the total accumulated pay for that week and then run tax calculations by working out a yearly income and then use that to calculate the tax for the week and medicare levy, based on which tax bracket that annual pay falls into, which is determined by and if/else clause to create the tax brackets.

```
    elif yearly_income > 29032 and yearly_income <= 45000:
        yearly_tax = (yearly_income - 18200) * .19
        weekly_tax = ((yearly_tax / 365) * 7) + weekly_medilevy
        weekly_tax = float(format(weekly_tax,".0f"))
        net_pay = weekly_accumulated - weekly_tax
        net_pay = format(net_pay,".2f")
        print(f"Your {fg('yellow')}gross weekly pay{attr('reset')} is: $",format(weekly_accumulated,".2f"))
        print(f"Your {fg('yellow')}net weekly pay{attr('reset')} is: ${net_pay} (Tax: {weekly_tax})")
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Weekly Total Gross: ", format(weekly_accumulated,".2f")])
            writer.writerow(["Weekly Total Net: ", net_pay])

    elif yearly_income > 45000 and yearly_income <= 120000:
        yearly_tax = (yearly_income - 45000) * .325
        weekly_static_tax = (5092 /365) * 7
        weekly_tax = ((yearly_tax / 365) * 7) + (weekly_medilevy + weekly_static_tax)
        weekly_tax = float(format(weekly_tax,".0f"))
        net_pay = weekly_accumulated - weekly_tax
        net_pay = format(net_pay,".2f")
        print(f"Your {fg('yellow')}gross weekly pay{attr('reset')} is: $",format(weekly_accumulated,".2f"))
        print(f"Your {fg('yellow')}net weekly pay{attr('reset')} is: ${net_pay} (Tax: {weekly_tax})")
        with open(file_name, "a")as pay_file:
            writer = csv.writer(pay_file)
            writer.writerow(["Weekly Total Gross: ", format(weekly_accumulated,".2f")])
            writer.writerow(["Weekly Total Net: ", net_pay])
```

```
elif yearly_income > 120000 and yearly_income <= 180000:
    yearly_tax = (yearly_income - 120000) * .37
    weekly_static_tax = (29467 /365) * 7
    weekly_tax = ((yearly_tax / 365) * 7) + (weekly_medilevy + weekly_static_tax)
    weekly_tax = float(format(weekly_tax,".0f"))
    net_pay = weekly_accumulated - weekly_tax
    net_pay = format(net_pay,".2f")
    print(f"Your {fg('yellow')}gross weekly pay{attr('reset')} is: $",format(weekly_accumulated,".2f"))
    print(f"Your {fg('yellow')}net weekly pay{attr('reset')} is: ${net_pay} (Tax: {weekly_tax})")
    with open(file_name, "a")as pay_file:
        writer = csv.writer(pay_file)
        writer.writerow(["Weekly Total Gross: ", format(weekly_accumulated,".2f")])
        writer.writerow(["Weekly Total Net: ", net_pay])

else:
    yearly_tax = (yearly_income - 180000) * .45
    weekly_static_tax = (51667 /365) * 7
    weekly_tax = ((yearly_tax / 365) * 7) + (weekly_medilevy + weekly_static_tax)
    weekly_tax = float(format(weekly_tax,".0f"))
    net_pay = weekly_accumulated - weekly_tax
    net_pay = format(net_pay,".2f")
    print(f"Your {fg('yellow')}gross weekly pay{attr('reset')} is: $",format(weekly_accumulated,".2f"))
    print(f"Your {fg('yellow')}net weekly pay{attr('reset')} is: ${net_pay} (Tax: {weekly_tax})")
    with open(file_name, "a")as pay_file:
        writer = csv.writer(pay_file)
        writer.writerow(["Weekly Total Gross: ", format(weekly_accumulated,".2f")])
        writer.writerow(["Weekly Total Net: ", net_pay])
```

Once the daily pay functions have all been run and the user inputs that they are finished the end of week pay calculations will run through the weekly_pay function. This takes the weekly accumulated pay, and converts that into an annual pay so that the function has a value to use for an if/else statement to find which tax bracket the user falls into so it can then calculate a weekly tax the user will need to pay then display these to the user in the app, and then write these values to the csv file. This is the code for the six separate tax brackets.

Terminal Application

Pay Calculator cont.

```
1. To calculate a new pay week  please enter 1
2. To view your pay history  please enter 2
3. To exit  please enter 3
Please enter your selection:  1
Please enter starting date of pay week in the format, DD-MM-YYYY : 17-04-2023
Please enter your base rate of pay : 36.52
Please enter a day you've worked, or 'finished' when your pay weeks complete : Monday
What shift did you work on Monday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Monday : 7.6
Was Monday a public holiday?  y/n.  n
Monday's pay is: $ 346.94
Weekly pay so far: $ 346.94
Please enter a day you've worked, or 'finished' when your pay weeks complete : Friday
What shift did you work on Friday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Friday : 7.6
Was Friday a public holiday?  y/n.  n
Friday's pay is: $ 398.07
Weekly pay so far: $ 745.01
Please enter a day you've worked, or 'finished' when your pay weeks complete : Saturday
What shift did you work on Saturday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Saturday : 7.6
Was Saturday a public holiday?  y/n.  n
Saturday's pay is: $ 477.68
Weekly pay so far: $ 1222.69
Please enter a day you've worked, or 'finished' when your pay weeks complete : Sunday
What shift did you work on Sunday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Sunday : 7.6
Was Sunday a public holiday?  y/n.  n
Sunday's pay is: $ 387.11
Weekly pay so far: $ 1609.80
Please enter a day you've worked, or 'finished' when your pay weeks complete : finished
Your gross weekly pay is: $ 1609.80
Your net weekly pay is: $1236.80 (Tax: 373.0)
Press enter to continue.....
```

```
1. To calculate a new pay week  please enter 1
2. To view your pay history  please enter 2
3. To exit  please enter 3
Please enter your selection:  1
Please enter starting date of pay week in the format, DD-MM-YYYY : 24-04-2023
Please enter your base rate of pay : 36.52
Please enter a day you've worked, or 'finished' when your pay weeks complete : Monday
What shift did you work on Monday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Monday : 7.6
Was Monday a public holiday?  y/n.  y
Monday's pay is: $ 693.88
Weekly pay so far: $ 693.88
Please enter a day you've worked, or 'finished' when your pay weeks complete : Friday
What shift did you work on Friday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Friday : 7.6
Was Friday a public holiday?  y/n.  n
Friday's pay is: $ 398.07
Weekly pay so far: $ 1091.95
Please enter a day you've worked, or 'finished' when your pay weeks complete : Saturday
What shift did you work on Saturday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Saturday : 7.6
Was Saturday a public holiday?  y/n.  n
Saturday's pay is: $ 477.68
Weekly pay so far: $ 1569.63
Please enter a day you've worked, or 'finished' when your pay weeks complete : Sunday
What shift did you work on Sunday? Please enter 'Day', 'Afternoon' or 'Night'.  Night
Please enter hours worked on Sunday : 7.6
Was Sunday a public holiday?  y/n.  n
Sunday's pay is: $ 387.11
Weekly pay so far: $ 1956.74
Please enter a day you've worked, or 'finished' when your pay weeks complete : finished
Your gross weekly pay is: $ 1956.74
Your net weekly pay is: $1464.74 (Tax: 492.0)
Press enter to continue.....
```

Here we have the full input and output for 2 pay weeks per the terminal application running the pay calculator, here someone who works 4 nights a week on night shift for 7.6 hours a day, with a pay week of monday, friday, saturday and sunday. In the second week on the monday nights there is a public holiday.

Here we have 2 whole pay weeks input by the user and outputs from the app. The one on the left is a 4 day week on night shift working monday, friday, saturday and sunday night. The one on the right is the same shifts but the monday night was a public holiday.

Terminal Application

View pay history

```
# Creates a csv file if one isnt already present
file_name = "pay_history.csv"
try:
    pay_file = open(file_name, "r")
    pay_file.close()
except FileNotFoundError:
    pay_file = open(file_name, "w")
    pay_file.write("pay_day,pay_amount\n")
    pay_file.close()
```

If the user chooses option number 2 to view their pay history then what they recieve will have been curated over the course of many other functions called through the pay calculator to input the information displayed in the pay history. First of all a csv file will be created if one isnt already present by the code to the left. The below code will add a date that the pay week starts when the user inputs it into the terminal as seen on the bottom of the screen.

```
# Pay week function to determine the starting date of the pay week, with try except to ensure correct date input, then writes to csv.
def pay_week(file_name):
    date_inputs = ''
    while True:
        date_inputs = input(f"Please enter starting date of pay week in the format, {fg('yellow')}DD-MM-YYYY{attr('reset')} : ")
        try:
            date_outputs = datetime.strptime(date_inputs, "%d-%m-%Y").date()
            with open(file_name, "a")as pay_file:
                writer = csv.writer(pay_file)
                writer.writerow(["Pay week: ", date_outputs])
            break
        except ValueError:
            print(f"{fg('red')}That is not a valid date.{attr('reset')}")
```

```
Please enter your selection:  1
Please enter starting date of pay week in the format, DD-MM-YYYY : 17-04-2023
```

When the app is first run it will try to open a csv file, if it cant it will then create one as per the code on the top left of this slide. When a user uses the pay calculator function they're asked to enter a date, this date is written to the csv file as seen in the code on the bottom of this slide.

Terminal Application

View pay history cont.

```
with open(file_name, "a") as pay_file:
    writer = csv.writer(pay_file)
    writer.writerow(["Monday", format(monday_pay,".2f")])
    return monday_pay
```

```
with open(file_name, "a") as pay_file:
    writer = csv.writer(pay_file)
    writer.writerow(["Weekly Total Gross: ", format(weekly_accumulated,".2f")])
    writer.writerow(["Weekly Total Net: ", net_pay])
```

As a day the user works gets entered into the pay calculator, each time the pay for that day is returned to the calculator function it will write that days pay to the csv file (above left). Once the user has finished their pay week and the full tax calculation has been completed, it will display it to the user and then write it to the csv file (above right). The code that allows the user to generate the pay history for viewing in the terminal application is below and the output they would receive for the 2 pay weeks entered into the app that we saw earlier would be the output on the right side of this slide.

```
# When the user enters option 2 this opens csv in read and displays for user.
def view_pay_history(file_name):
    print(f"{fg('yellow')}Viewing pay history{attr('reset')}")
    with open(file_name, "r") as pay_file:
        reader = csv.reader(pay_file)
        for row in reader:
            print(row)
```

```
1. To calculate a new pay week  please enter 1
2. To view your pay history  please enter 2
3. To exit  please enter 3
Please enter your selection:  2
Viewing pay history
['pay_day', 'pay_amount']
['Pay week: ', '2023-04-17']
['Monday', '346.94']
['Friday', '398.07']
['Saturday', '477.68']
['Sunday', '387.11']
['Weekly Total Gross: ', '1609.80']
['Weekly Total Net: ', '1236.80']
['Pay week: ', '2023-04-24']
['Monday', '693.88']
['Friday', '398.07']
['Saturday', '477.68']
['Sunday', '387.11']
['Weekly Total Gross: ', '1956.74']
['Weekly Total Net: ', '1464.74']
Press enter to continue.....
1. To calculate a new pay week  please enter 1
2. To view your pay history  please enter 2
3. To exit  please enter 3
Please enter your selection:  3
You are now exiting.
Thank you for using the pay calculator!
```

Similar to this is when the user daily pay is returned and the weekly calculations are done they are also written to the csv file, forming the whole history for a pay week at the end of the first option 1 pay calculator. Now if the user inputs 2 on the first main menu, to view their pay history the code at the bottom is run and the csv file is opened in read mode and displayed in the app as per the display on the right. Once the user has viewed their history and wants to exit the app they can press enter to continue and then 3 to exit.

Terminal Application

Testing

```python
# Testing that each of the pay rate functions returns the right value
def test_basic():
    assert "hello world" == "hello world"

def test_ph_pay():
    assert public_holiday_pay(50, 10) == 1250

def test_ds_pay():
    assert day_shift_pay(50, 10) == 500

def test_ns_pay():
    assert night_shift_pay(50, 10) == 625

def test_sat_pay():
    assert sat_loading_pay(50, 10) == 750

def test_sun_pay():
    assert sun_loading_pay(50, 10) == 900
```

```python
def test_add(monkeypatch):
    original_length = 0
    with open(test_file_name) as f:
        reader = csv.reader(f)
        original_length = sum(1 for row in reader)
    monkeypatch.setattr('builtins.input', lambda _: "01-01-0101")
    pay_week(test_file_name)
    with open(test_file_name) as f:
        reader = csv.reader(f)
        new_length = sum(1 for row in reader)
    print(original_length)
    print(new_length)
    assert new_length == original_length + 1
```

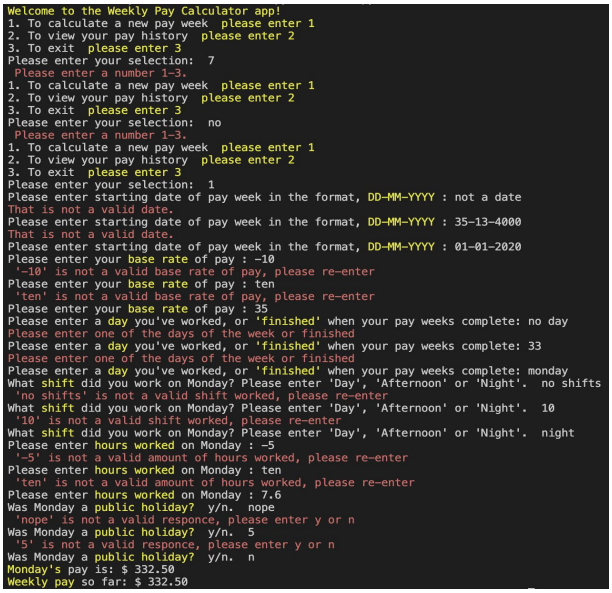
```
============================= test session starts ==============================
platform darwin -- Python 3.9.6, pytest-7.3.1, pluggy-1.0.0
rootdir: /Users/scott.taylor/Documents/WebDevTerm1/T1A3-Terminal_Application/ScottTaylor_T1A3/src
collected 7 items

test_ta.py .......                                                       [100%]

============================== 7 passed in 0.01s ===============================
```

With the installation of an external package called pytest, some tests can be performed that will determine if functions used in the application are working as intended. I chose to run test on the pay functions that calculate the daily pay for the user, and to test if when the user enters a date into the pay calculator if it adds a new line to the csv file. All tests passed.

Now to testing the application, this was done with pytest the python external package and as you can see in the code here we have tested the calculation functions to ascertain that they return the correct values and then a test to see if the pay week function is adding a line to the csv file when its supposed to. The output at the bottom shows that all these tests pass.

Terminal Application

Error Handling



As seen in many of the code snippets, any input by the user needs to have provisions in place that ensure only acceptable input go into the application to avoid any crashing of the program due to errors of exception. Some have string rejection through else statements if string matching isn't met, others have their own functions that have either try/except blocks for value or type errors with manual raises of exception, or their own if/else matching to break or continue the loops depending on user input.

Another form of testing was to ensure that there were no errors that could occur in the application. This was done a few different ways with string matching, try and except blocks and manual raising of exceptions to ensure no matter what gets entered the application continues on. As can be seen on the left of the screen, everything in red is a message to the user that their input was not correct and to re enter.

Terminal Application

Development Process

The development process was really just a matter of breaking down each part of the code. Started by setting up the main menu and creating the try/except for the creation of the CSV file. Once the menu was set up, needed to move onto the pay calculator function that was the major part of the application. Once setting up a way to loop through all of the days and to end the loop when done, it was time to start creating functions for getting inputs and then running calculations on those inputs to get a figure for each days pay, creating a sum of each day as the loop rolls through the days until the week was over. Next came the hard part of creating a function that could take that and then determine tax brackets and tax that needed to be paid, this wasn't originally in the development plan but was something in the back of my mind that I wanted to do but wasn't sure how to go about making it work, but eventually got it sorted. After that came figuring out how to make wrong inputs not crash the program by making majority of inputs their own functions so that I could loop over the answer until a correct one was given. Setting up the pytest package to test some functions was next on the list and then finally setting up a bash script that can be used as an executable.

The development process was really just trying to work on one function at a time, make sure it worked then move onto the next one. Then test the application with pytest, and make a bash script for the user, pushing commits to github whenever something new was added to the application. Thanks for watching my presentation.