// main.py

```python
# Portions of this code were developed with assistance from an AI tool (Claude)
from datetime import datetime
import json
import requests
from colorama import Fore, Style
from menu import InteractiveMenu

SUPPORTED_CURRENCIES = [
    "USD", "EUR", "GBP", "JPY", "AUD", "CAD", "CHF", "CNY",
    "SEK", "NZD", "MXN", "SGD", "HKD", "NOK", "KRW", "TRY",
    "INR", "BRL", "ZAR", "DKK", "PLN", "THB", "IDR", "HUF",
    "CZK", "ILS", "PHP", "MYR", "RON", "BGN", "ISK",
]

class ExpenseTracker():
    def __init__(self,filename='data.txt'):
        self.filename = filename

    def open_file(self) -> list:
        try:
            with open(self.filename,'r') as file:
                expenseList = json.load(file)
            return expenseList
        except FileNotFoundError:
            print("File doesn't exist, creating ...")
            with open(self.filename,'w') as file:
                json.dump([],file)
            return []

    def write_file(self,data:list):
        try:
            with open(self.filename,'w') as file:
                json.dump(data,file)
        except FileNotFoundError:
            with open(self.filename,'w') as file:
                json.dump([],file)

    def assign_id(self) -> int:
        totalExpenseList = self.open_file()
        if len(totalExpenseList) == 0:
            return 1
        else:
            max_id = max(expense['id'] for expense in totalExpenseList)
            return max_id + 1

    def convert_currency(self,price:float,from_curr:str,to_curr:str) -> float:
        from_curr = from_curr.upper()
        to_curr = to_curr.upper()
        if from_curr == to_curr:
            return price
        url = f"https://api.frankfurter.app/latest?from={from_curr}&to={to_curr}"
        response = requests.get(url)
        data = response.json()
        if 'rates' not in data or to_curr not in data['rates']:
            raise ValueError(f"Could not convert from {from_curr} to {to_curr}: {data.get('message', 'unknown error')}")
        return price * data['rates'][to_curr]

    def view_total_expenses(self)-> list:
        expenseList = self.open_file()
        return expenseList

    def view_filtered_expenses(self)-> list:
        expenseList = self.open_file()
        if not expenseList:
            return []
        filter_menu = InteractiveMenu(
            ["By Price", "By Item", "By Date"],
            title="Filter By"
        )
        filter_choice = filter_menu.show()
        if filter_choice is None:
            return []
        filteredExpenses = []
        if filter_choice == "By Price":
            filter_min_value = float(input("Enter the minimum value:\n> "))
            filter_max_value = float(input("Enter the maximum value:\n> "))
            for expense in expenseList:
                if filter_min_value <= expense['price'] <= filter_max_value:
                    filteredExpenses.append(expense)
        elif filter_choice == "By Item":
            filter_item = input("Search for item:\n> ").lower()
            for expense in expenseList:
                if filter_item in expense['purchased'].lower():
                    filteredExpenses.append(expense)
        elif filter_choice == "By Date":
            filter_min_date = input("Enter the start range (yyyy-mm-dd):\n> ")
            filter_max_date = input("Enter the end range (yyyy-mm-dd):\n> ")
            for expense in expenseList:
                if filter_min_date <= expense['date'] <= filter_max_date:
                    filteredExpenses.append(expense)
        return filteredExpenses

    def add_expenses(self,price:float,purchased:str,currency:str='usd',date:str='',notes:str='')-> str:
        if not date:
            date = datetime.now().strftime("%Y-%m-%d")
        expense = {
            'id': self.assign_id(),
            'price': price,
            'purchased': purchased,
            'date': date,
            'currency': currency.lower(),
            'notes': notes,
        }
        expenseList = self.open_file()
        expenseList.append(expense)
        self.write_file(expenseList)
        return "Expense properly added"

    def edit_expenses(self)-> str:
        expenseList = self.open_file()
        if not expenseList:
            return "No expenses to edit."
        options = [f"{e['id']}: {e['purchased']} - {e['price']} {e['currency'].upper()} ({e['date']})" for e in expenseList
        select_menu = InteractiveMenu(options, title="Select Expense")
        selected = select_menu.show()
        if selected is None:
            return "Edit cancelled."
        selected_id = int(selected.split(":")[0])
        edit_menu = InteractiveMenu(
            ["Price", "Purchased Item", "Date of Purchase", "Notes"],
            title="Edit Field"
        )
        choice = edit_menu.show()
        if choice is None:
            return "Edit cancelled."
        for expense in expenseList:
            if expense['id'] == selected_id:
                if choice == "Price":
                    new_price = float(input("Enter the new price:\n> "))
                    expense['price'] = new_price
                elif choice == "Purchased Item":
                    new_purchased = str(input("Enter the new purchased item:\n> "))
                    expense['purchased'] = new_purchased
                elif choice == "Date of Purchase":
                    new_date = str(input("Enter the new date (yyyy-mm-dd):\n> "))
                    expense['date'] = new_date
                elif choice == "Notes":
                    new_notes = input("Enter the new notes:\n> ")
                    expense['notes'] = new_notes
        self.write_file(expenseList)
        return "Expense edited successfully."

    def delete_expenses(self)-> str:
        expenseList = self.open_file()
        if not expenseList:
            return "No expenses to delete."
        options = [f"{e['id']}: {e['purchased']} - {e['price']} {e['currency'].upper()} ({e['date']})" for e in expenseList
        select_menu = InteractiveMenu(options, title="Delete Expense")
        selected = select_menu.show()
        if selected is None:
            return "Delete cancelled."
        selected_id = int(selected.split(":")[0])
        expenseList = [expense for expense in expenseList if expense['id'] != selected_id]
        self.write_file(expenseList)
        return "Expense deleted successfully."

    def export_to_csv(self,filename='expenses.csv')-> str:
        expenseList = self.open_file()
        if not expenseList:
            return "No expenses to process."
        with open(filename,'w') as file:
            file.write("id,price,purchased,date,currency,notes\n")
            for expense in expenseList:
                notes = expense.get('notes', '')
                file.write(f"{expense['id']},{expense['price']},{expense['purchased']},{expense['date']},{expense['currency
        return "Expenses exported successfully."

    def convert_prices_to_currency(self,to_currency:str)-> str:
        expenseList = self.open_file()
        if not expenseList:
            return "No expenses to convert."
        try:
            for expense in expenseList:
                from_currency = expense['currency']
                price_in_new_currency = self.convert_currency(expense['price'],from_currency,to_currency)
                expense['price'] = round(float(price_in_new_currency),2)
                expense['currency'] = to_currency.lower()
            self.write_file(expenseList)
            return "All expenses converted successfully."
        except ValueError as e:
            return str(e)

def format_expenses(expenses: list) -> str:
    if not expenses:
        return f"{Fore.YELLOW}No expenses found.{Style.RESET_ALL}"
    header = (
        f"{Fore.CYAN}{Style.BRIGHT}"
        f"{'ID':<6}{'Item':<20}{'Price':>10}  {'Curr':<6}{'Date':<12}{'Notes'}"
        f"{Style.RESET_ALL}"
    )
    separator = f"{Fore.CYAN}{'?' * 78}{Style.RESET_ALL}"
    lines = [separator, header, separator]
    total = 0.0
    for e in expenses:
        price = float(e['price'])
        total += price
        notes = e.get('notes', '')
        if len(notes) > 20:
            notes = notes[:17] + "..."
        lines.append(
            f"{e['id']:<6}{e['purchased']:<20}{price:>10.2f}  {e['currency'].upper():<6}{e['date']:<12}{Fore.YELLOW}{notes}
        )
    lines.append(separator)
    lines.append(
        f"{Style.BRIGHT}{'Total':<26}{total:>10.2f}{Style.RESET_ALL}"
    )
    lines.append(separator)
    return "\n".join(lines)

tracker = ExpenseTracker()
menu_options = [
    "View total expenses",
    "Filter total expenses",
    "Add expenses",
    "Edit expenses",
    "Delete expenses",
    "Export expenses to CSV",
    "Convert currency",
    "Exit",
]

running = True
while running:
    menu = InteractiveMenu(menu_options, title="Expense Tracker")
    choice = menu.show()

    if choice == "View total expenses":
        print(format_expenses(tracker.view_total_expenses()))
    elif choice == "Filter total expenses":
        print(format_expenses(tracker.view_filtered_expenses()))
    elif choice == "Add expenses":
        price = float(input("How much was spent?\n> "))
        purchased = input("What was purchased?\n> ")
        notes = input("Notes (optional):\n> ")
        currency_menu = InteractiveMenu(SUPPORTED_CURRENCIES, title="Currency")
        currency = currency_menu.show()
        if currency is None:
            print("Add cancelled.")
        else:
            print(tracker.add_expenses(price, purchased, currency, notes=notes))
    elif choice == "Edit expenses":
        print(tracker.edit_expenses())
    elif choice == "Delete expenses":
        print(tracker.delete_expenses())
    elif choice == "Export expenses to CSV":
        print(tracker.export_to_csv())
    elif choice == "Convert currency":
        currency_menu = InteractiveMenu(SUPPORTED_CURRENCIES, title="Convert To")
        to_currency = currency_menu.show()
        if to_currency is None:
            print("Conversion cancelled.")
        else:
            print(tracker.convert_prices_to_currency(to_currency))
    elif choice == "Exit" or choice is None:
        running = False

    if running:
        input("Press Enter to continue...")

```

// menu.py

```python
import sys
import os
from colorama import Fore, Style, init
from math import ceil
from typing import List, Optional


class InteractiveMenu:
    """
    A responsive interactive menu system with keyboard navigation.
    Supports Windows (msvcrt) and macOS/Linux (tty/termios).
    """

    def __init__(self,
                 options: List[str],
                 cursor: str = "?",
                 highlight_color: str = Fore.GREEN,
                 items_per_page: int = 10,
                 title: str = "Menu"):
        """Initialize the interactive menu."""
        init(convert=True)
        self.original_options = [opt for opt in options if opt.strip()]
        self.cursor = cursor
        self.highlight_color = highlight_color
        self.items_per_page = items_per_page
        self.title = title
        self.current_index = 0
        self.current_page = 1
        self.total_pages = max(1, ceil(len(self.original_options) / items_per_page))

        self.can_navigate = False
        self._platform = None
        self.msvcrt = None
        self.tty = None
        self.termios = None
        if os.name == 'nt':
            try:
                import msvcrt
                self.msvcrt = msvcrt
                self.can_navigate = True
                self._platform = 'windows'
            except ImportError:
                pass
        else:
            try:
                import tty
                import termios
                self.tty = tty
                self.termios = termios
                self.can_navigate = True
                self._platform = 'unix'
            except ImportError:
                pass

    def _clear_screen(self) -> None:
        """Clear the terminal screen."""
        os.system('cls' if os.name == 'nt' else 'clear')

    def _get_key(self) -> Optional[str]:
        """Get a single keypress without Enter. Supports Windows and Unix."""
        if not self.can_navigate:
            return None

        if self._platform == 'windows' and self.msvcrt is not None:
            if self.msvcrt.kbhit():
                key = self.msvcrt.getch()
                if key == b'\xe0':  # Special key prefix on Windows
                    key = self.msvcrt.getch()
                    if key == b'H': return 'up'
                    elif key == b'P': return 'down'
                    elif key == b'K': return 'left'
                    elif key == b'M': return 'right'
                elif key == b'\r': return 'enter'
                elif key == b'\x1b': return 'escape'
                elif key.lower() == b'r': return 'reset'
                elif key.lower() == b'q': return 'quit'
        elif self._platform == 'unix':
            ch = sys.stdin.read(1)
            if ch == '\x1b':  # Escape sequence
                seq = sys.stdin.read(2)
                if seq == '[A': return 'up'
                elif seq == '[B': return 'down'
                elif seq == '[C': return 'right'
                elif seq == '[D': return 'left'
                return 'escape'
            elif ch in ('\r', '\n'): return 'enter'
            elif ch.lower() == 'r': return 'reset'
            elif ch.lower() == 'q': return 'quit'
        return None

    def _get_page_options(self) -> List[str]:
        """Get options for the current page."""
        if len(self.original_options) <= self.items_per_page:
            return self.original_options

        start_idx = (self.current_page - 1) * self.items_per_page
        end_idx = min(start_idx + self.items_per_page, len(self.original_options))
        return self.original_options[start_idx:end_idx]

    def _get_display_index(self) -> int:
        """Get the display index for the current page."""
        if len(self.original_options) <= self.items_per_page:
            return self.current_index
        return self.current_index % self.items_per_page

    def _render_menu(self) -> None:
        """Render the menu to the terminal."""
        # Header
        print(f"{'?' * 20} {self.title} {'?' * 20}")

        # Page info for multi-page menus
        if self.total_pages > 1:
            print(f"Page [{self.current_page}/{self.total_pages}]")

        # Instructions
        if self.can_navigate:
            print("Use ?? arrow keys to navigate | ENTER to select | ESC/Q to cancel | R to reset")
            if self.total_pages > 1:
                print("Use ?? arrow keys to change pages")
        else:
            print("Interactive navigation not available.")

        print()  # Empty line before options

        # Render options
        page_options = self._get_page_options()
        display_index = self._get_display_index()

        for i, option in enumerate(page_options):
            if i == display_index:
                print(f"{self.highlight_color}{Style.BRIGHT}{self.cursor} {option}{Style.RESET_ALL}")
            else:
                print(f"  {option}")

        print()  # Empty line after options
        sys.stdout.flush()

    def show(self) -> Optional[str]:
        """Display the menu and return the selected option."""
        if not self.original_options:
            print("No options provided!")
            return None
        if not self.can_navigate:
            print("Interactive navigation not available.")
            return None

        old_settings = None
        if self._platform == 'unix' and self.termios is not None and self.tty is not None:
            old_settings = self.termios.tcgetattr(sys.stdin)
            self.tty.setcbreak(sys.stdin.fileno())

        try:
            # Initial render
            self._clear_screen()
            self._render_menu()

            while True:
                key = self._get_key()
                if key is None:
                    continue

                needs_refresh = False

                if key == 'up' and self.current_index > 0:
                    self.current_index -= 1
                    new_page = (self.current_index // self.items_per_page) + 1
                    if new_page != self.current_page:
                        self.current_page = new_page
                    needs_refresh = True

                elif key == 'down' and self.current_index < len(self.original_options) - 1:
                    self.current_index += 1
                    new_page = (self.current_index // self.items_per_page) + 1
                    if new_page != self.current_page:
                        self.current_page = new_page
                    needs_refresh = True

                elif key == 'left' and self.total_pages > 1 and self.current_page > 1:
                    self.current_page -= 1
                    self.current_index = (self.current_page - 1) * self.items_per_page
                    needs_refresh = True

                elif key == 'right' and self.total_pages > 1 and self.current_page < self.total_pages:
                    self.current_page += 1
                    self.current_index = min(
                        (self.current_page - 1) * self.items_per_page,
                        len(self.original_options) - 1
                    )
                    needs_refresh = True

                elif key == 'reset':
                    self.current_index = 0
                    self.current_page = 1
                    needs_refresh = True

                elif key == 'enter':
                    selected_option = self.original_options[self.current_index]
                    self._clear_screen()
                    return selected_option

                elif key in ['escape', 'quit']:
                    self._clear_screen()
                    return None

                # Only refresh if something changed
                if needs_refresh:
                    self._clear_screen()
                    self._render_menu()

        except KeyboardInterrupt:
            self._clear_screen()
            return None
        except Exception as e:
```

```python
            self._clear_screen()
            print(f"Error: {e}")
            return None
        finally:
            if old_settings is not None and self.termios is not None:
                self.termios.tcsetattr(sys.stdin, self.termios.TCSADRAIN, old_settings)

if __name__ == '__main__':
    options = ["Test1", "Test2", "Balls"] + [f"Item {i}" for i in range(20)]
    menu = InteractiveMenu(options)
    result = menu.show()
    print(result)
```