import data

def display\_dict(a\_dict, label=None, \*\*kwargs):

"""

Displays a dictionary formatted nicely.

"""

if label:

print(f"{label}:")

for key, value in a\_dict.items():

print(f"{key:>25}: {value}")

def get\_item\_from\_list(the\_list):

"""

Presents user with list of choices until they select an item from it.

"""

valid\_choices = range(1, len(the\_list)+1)

while True:

# Display choices and get input

print("Select an item:")

for item\_label, item in enumerate(the\_list, start=1):

print(f"{item\_label}: {item}")

selected\_label = input(">>>> ").strip()

try:

# Index is one less than user's choice integer

if int(selected\_label) not in valid\_choices:

# User's input isn't a valid choice

print(f"Choice must be between 1 and {len(the\_list)}")

continue

else:

selected\_index = int(selected\_label.strip())-1

# User's input is valid

break

except:

# User's input can't be converted to an integer

print("Invalid choice")

continue

# Return selected item from list

return the\_list[selected\_index]

def get\_column\_type\_dict(sample\_row, column\_type\_dict, types=[str, int, float, bool], \*\*kwargs):

"""

Given a dictionary of CSV row data, return a dict of column names with

user-selected data types.

For each column, present user with sample data. Get selected data type for each

until conversion is successful. Build dict of column names and data types.

Returns dict with `column\_type\_dict` key and resulting dict as value.

Assumes sample\_row is representative of all data in CSV.

"""

display\_dict(column\_type\_dict, label="Current column data types:")

data\_type\_dict = {}

columns = sample\_row.keys()

for column in columns:

while True:

data = sample\_row[column]

print(f"What data type is {column}?\n Sample data:\n {data}\n")

selected\_type = get\_item\_from\_list(types)

try:

selected\_type(data)

break

except:

# data could not be converted to that type

print("Data could not be converted to that type. Make another selection.")

types.remove(selected\_type)

data\_type\_dict[column] = selected\_type

return {

"column\_type\_dict" : data\_type\_dict

}

def get\_valid\_row\_index(row\_list, default = 1, prompt="Row to view"):

"""

Gets a valid row\_int got a given row\_list

"""

# Find highest row number

maxrow\_int = len(row\_list)

while True:

print(f"Enter a row number from 1 to {maxrow\_int}")

row\_choice\_str = input(f"{prompt} (default: {default}): ")

if row\_choice\_str == "":

# User just pressed enter; use default

row\_int = default

break

try:

# User entered something; try to convert it to an integer

row\_int = int(row\_choice\_str)

except:

print("Please enter an integer")

continue # User must start over; repeat loop

# Make sure integer is a valid row number

if row\_int < 1 or row\_int > maxrow\_int:

print("Not a valid row number")

continue # User must start over; repeat loop

else:

break

# The list index will be one less than the user-selected row number

row\_index = row\_int -1

return row\_index

def input\_is\_yes(message\_str, default = None):

"""

Displays `message` and collects validated yes/y/no/n input.

Returns `True` for yes answers, `False` for no answers.

Optional `default` parameter can be any yes or no value; if set,

empty string acts as that answer. Also affects option

formatting. 'y' or 'n' recommended.

"""

yes\_options\_list = ["y", "yes","ok","yea","yeah"]

no\_options\_list = ["n", "no", "nope", "nah"]

assert default in yes\_options\_list + no\_options\_list + [None], "`default` argument must be a form of yes, no, or None."

if default in yes\_options\_list:

options\_str = f"({default.upper()}/n)"

yes\_options\_list.append("")

elif default in no\_options\_list:

options\_str = f"(y/{default.upper()})"

no\_options\_list.append("")

while True:

choice\_str = input(f'{message\_str} {options\_str} ').strip().lower()

if choice\_str in yes\_options\_list:

return True

elif choice\_str in no\_options\_list:

return False

else:

print(f"{choice\_str} is not valid input")

continue

def present\_choices(choices\_dict):

"""

Takes a dict with choice string keys and function dict

values. Expects function dicts to have "description"

and "function" keys/.

Present choices to the user and return the

corresponding function name.

"""

while True:

print("Main Menu:")

choices\_list = sorted(choices\_dict.keys())

for choice in choices\_list:

function\_dict = choices\_dict[choice]

description = function\_dict["description"]

print(f"{choice}: {description}")

choice\_str = input("Make a selection: ")

if choice\_str in choices\_dict:

function\_dict = choices\_dict[choice\_str]

function = function\_dict["function"]

return function

elif choice\_str == "exit":

print("Goodbye")

exit()

else:

print("Invalid selection")

continue

def choose\_csv():

"""

Prints a list of CSVs visible to Python and returns the filename string

the user selects from the list.

User must enter an integer between one and the number of CSVs in the list

of CSV returned by `data.get\_csvs\_list()`

"""

new\_filename\_str = ""

csvs\_list = data.get\_csv\_filenames\_list()

while new\_filename\_str not in csvs\_list:

print("Available CSV files:")

for i, filename in enumerate(csvs\_list, start = 1):

print(f"{i} - {filename}")

try:

filenum = int(input(f"Enter a number between 1 and {i}: "))

new\_filename\_str = csvs\_list[filenum-1]

print("Chosen CSV file:",new\_filename\_str, end="\n\n")

except:

# Can't understand user input. Ask again.

print("Try again.")

continue

return new\_filename\_str