Project Final Submission Template

Step 1a: Planning

Identify the information in the file your program will read

FROM THE UNHDI FILE:

- 2018 HDI ranking (how a countries HDI compares to others)
- Country name (excludes many self-declared states such as Taiwan, Somaliland, etc.)
- The years from 1990-2018
- Each countries respecitve HDI for a particular year
- HDIs considered Very High, High, Medium, Low, and Developing in each year
- The average HDI of different regions (East Asia, Europe, Arab States, etc.)
- The average HDI of the world

FROM THE USAID GREENBOOK:

- The fiscal year
- The region
- The country name
- The assistance category (military or economic)
- The publication row (type of financial aid)
- The funding agency (either the department of State or Defence)
- The funding account name
- The amount of money spent in USD (as its worth in that year)
- The amount of money spent in USD (adjusted for inflation as of 2018)

Step 1b: Planning

line chart: (i was originally going to do this, as per my project proposal, but the computation didn't seem substantial :( )

- Title: Percent Change in the Human Development Index of Countries Funded By The United States Military from 1990 to Present
- y axis: The % change of the HDI from the year previous of countries
    - The only countries shown will be those occupied by the United States Military from 1990 to present
- x-axis: time in years
- Each country will be plotted in a separate line and colour
- There will be symbols showing when occupation began and ended
    - The types of symbols (circle, square, triangle, etc.) will show what type of occupation it was (drone strikes, military training, combat, etc.)
- On the side, there will be a figure legend describing the symbols (occupation type) as well as the colours (country)

bar graph:

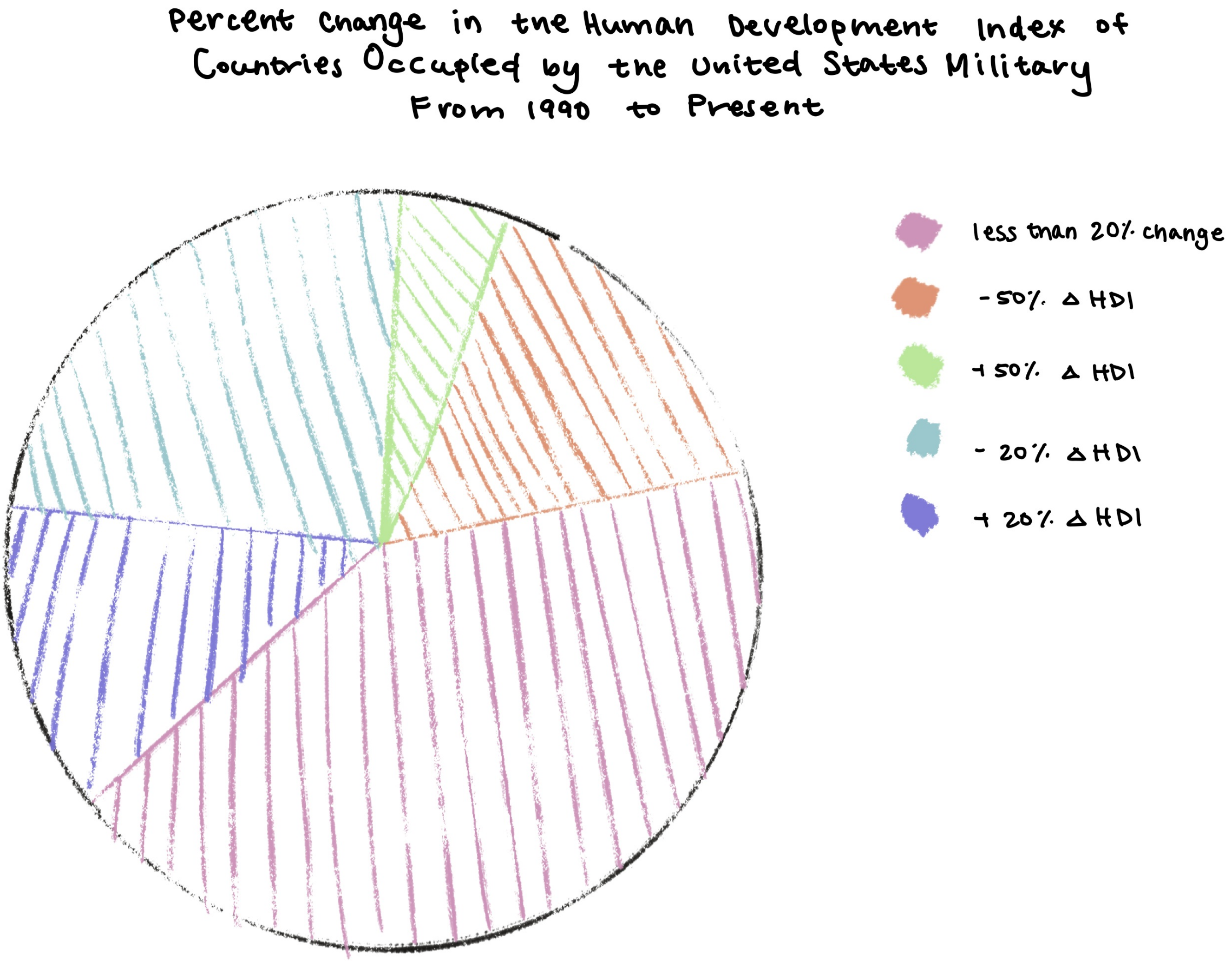
- Title: Average Human Development Index of Countries Funded By The United States Military from 1990 to Present
- There will be four bars, split into two subsets:
    - One subset will be of >10 years of occupation, and the other will show <10 years of occupation
    - Within the subsets:
        - One bar will be the average HDI of countries before there was United States military occupation
        - The other bar will be the average HDI after US occupation
- The averages will be taken at the beginning & end of occupation, regardless of year.
- There will be subtitles below each bar to indicate averages before and after occupation
- Below the figure, there will be a figure legend mentioning which countries contributed to which bar

pie chart: (the one i'm choosing!!)

- Title: Percent Change in the Human Development Index of Countries Funded By The United States Military from 1990 to Present
- The chart will be split into 5 colours:
    - countries whose % change from before and after was >= 50%
    - countries whose % change before and after occupation was <= -50%
    - countries whose % change from before and after was >= 20%
    - countries whose % change from before and after was <= -20%
    - countries whose % change hadn't changed much (< |+/-20%| change over the course of occupation.
- These percentage 'cutoffs' are subject to change, as they were abitrarily chosen. It really depends on the data and how best to represent the disparities
- On the side, there will be a figure legend describing the colours (% changes)
- Below the figure, there will be a figure legend mentioning which countries contributed to which pie "slice"

Step 1c: Planning

```
expect(main("Human Developement Index (HDI).csv", "us_foreignaid_greenbook.csv"), None)
# see below for the expected outcome
```



Step 2a: Building

Document which information you will represent in your data definitions

- The Human Development Index of each year `(float)`
  - the year itself is unnessecary because each list will have 29 values coresponding to 29 years
  - unknown HDIs will simply have None
- The name of the country `(str)`

These are the only ones that are needed, since:

1. We'll be doing calculations on the HDI and plotting it against time
   - once again, time is not relevant to extract from the file because it is implied in the data already stored
2. We need country names to filter for the countries that have been affected by the issue i'd like to address

Data Definitions

In [41]:

```
from cs103 import *
from typing import NamedTuple, List, Optional
from enum import Enum
import csv

#################
# for UNHDI
```

```
HumanDevelopmentIndex = Optional[float] # in range [0.0, 1.0]
# interp. a countries Human Development Index from the United Nations. None means the HDI is unknown.

HDI1 = None
HDI2 = 0.923
HDI3 = 0.293

# template from: One Of (2 cases), Atomic Distinct (1 case), and  Atomic Non-Distinct (1 case)
@typecheck
def fn_for_human_development_index(hdi: HumanDevelopmentIndex) -> ...:
    if hdi is None:
        return ...
    else:
        return ...(hdi)



# List[HumanDevelopmentIndex]
# interp. a list of Human Development Indexes

LOHDI1 = []
LOHDI2 = [None]
LOHDI3 = [None, .324, .390]
LOHDI4 = [0.893, 0.900, 0.931]

# template from Arbitraty-Sized and the Reference Rule
@typecheck
def fn_for_list_of_hdi(lohdi: List[HumanDevelopmentIndex]) ->...:
    # description of the accumulator
    acc = ...      # type: ...
    for hdi in lohdi:
        acc = ...(fn_for_human_development_index(hdi), acc)
    return ...(acc)



UNCountry = NamedTuple('UNCountry', [('name', str),
                                ('HDI', List[HumanDevelopmentIndex])])
# interp. a countries' name ('name'), and the list of United Namtions Human Development Indexes ('HDI') recorded
#         from 1990 to 2018. Assume no list in HDI is empty, but it can have None as all the values.


UNC1 = UNCountry('Afghanistan', [0.298, 0.304, 0.312, 0.308, 0.303, 0.327, 0.331, 0.335, 0.339, 0.343, 0.345,
                             0.347, 0.378, 0.387,0.4, 0.41, 0.419, 0.431, 0.436, 0.447, 0.464, 0.465, 0.479,
                             0.485, 0.488, 0.49, 0.491, 0.493, 0.496])
UNC2 = UNCountry('Bangladesh', [0.388, 0.395, 0.403, 0.411, 0.419, 0.427, 0.436, 0.444, 0.453, 0.462, 0.47,
                            0.479, 0.485, 0.492, 0.499, 0.506, 0.514, 0.521, 0.524, 0.535, 0.549, 0.559,
                            0.567, 0.572, 0.572, 0.588, 0.599, 0.609, 0.614])
UNC3 = UNCountry('Andorra', [None, None, None, None, None, None, None, None, None, None, 0.759, 0.767, 0.78,
                         0.82, 0.826, 0.819, 0.829, 0.829, 0.831, 0.83, 0.828, 0.827, 0.849, 0.846, 0.853,
                         0.85, 0.854, 0.852, 0.857])
UNC4 = UNCountry("Korea (Democratic People's Rep. of)", [None, None, None, None, None, None, None, None, None,
                                                     None, None, None, None, None, None, None, None, None, None,
                                                     None, None, None, None, None, None, None, None, None, None])

# template from Compound and the Reference Rule
@typecheck
def fn_for_uncountry(unc: UNCountry) -> ...:
    return ...(unc.name,                    # str
              fn_for_list_of_hdi(unc.HDI))  # HumanDevelopmentIndex



# List[UNCountry]
# interp. a list of UNCountry

LOUNC0 = []
LOUNC1 = [UNC2, UNC3, UNC4]

# template from Arbitraty-Sized and the Reference Rule
@typecheck
def fn_for_lounc(lounc: List[UNCountry]) -> ...:
    # description of the accumulator
    acc = ...      # type: ...
    for unc in lounc:
        acc = ...(fn_for_uncountry(unc), acc)
    return ...(acc)

##################
# for USAID

USCountry = NamedTuple('USCountry', [('year', int), # in range [1990, 2018]
                                     ('name', str),
                                     ('aid', str),

                                     ('amt', float)]) # in range [0, ...]
# interp. a countries' name ('name'), the year the US gave their military financial aid from 1990 to 2018, and
#         the financial aid type ('aid'), and the dollar amount ('amt')

USC1 = USCountry(1990, 'Algeria', 'Military', 144000000.00)
USC2 = USCountry(1990, 'Egypt', 'Economic', 898389000.00)

@typecheck
# template from Compound
def fn_for_uscountry(usc: USCountry) -> ...:
    return ... (usc.year, # int in range [1990, 2018]
               usc.name,  # str
               usc.aid,   # str
               usc.amt)   # int in range [0, ...]

# List[USCountry]
# interp. a list of USCountry

LOUSC0 = []
LOUSC1 = [USC1, USC2]

# template from Arbitraty-Sized and the Reference Rule
@typecheck
def fn_for_lousc(lousc: List[USCountry]) -> ...:
    # description of the accumulator
    acc = ...      # type: ...
    for usc in lousc:
        acc = ...(fn_for_uscountry(usc), acc)
    return ...(acc)
```

Step 2b and 2c: Building

Design a function to read the information and store it as data in your program

Design functions to analyze the data

Complete these steps in the code cell below. You will likely want to rename the analyze function so that the function name describes what your analysis function does.

In [42]:

```
###########
# functions for main:

#@typecheck
#def main(filename: str) -> ...:
 #   """
 #   Reads the file from given filename, analyzes the data, returns the result
  #  """
    # Template from HtDAP, based on function composition
    #return analyze(read(filename))


start_testing()

# Examples and tests for main
# expect(main('Human Development Index (HDI) test 1.csv'), None)
#expect(main('Human Development Index (HDI) test 2.csv'), None)

summary()
```

```
0 of 0 tests passed
```

functions for read:

In [40]:

```
@typecheck
def read_unc(filename: str) -> List[UNCountry]:
    """
    reads information from the specified file and returns a list of United Nations HDI country data
    """
    #return []  #stub
    # Template from HtDAP
    # loc contains the result so far
    lounc = [] # type: List[UNCountry]

    with open(filename) as csvfile:

        reader = csv.reader(csvfile)
        next(reader) # skip first header line
        next(reader) # skip second header line

        for row in reader:
            unc = UNCountry(row[1], [parse_float(row[2]), parse_float(row[3]), parse_float(row[4]),
                                     parse_float(row[5]), parse_float(row[6]), parse_float(row[7]),
                                     parse_float(row[8]), parse_float(row[9]), parse_float(row[10]),
                                     parse_float(row[11]), parse_float(row[12]), parse_float(row[13]),
                                     parse_float(row[14]), parse_float(row[15]), parse_float(row[16]),
                                     parse_float(row[17]), parse_float(row[18]), parse_float(row[19]),
                                     parse_float(row[20]), parse_float(row[21]), parse_float(row[22]),
                                     parse_float(row[23]), parse_float(row[24]), parse_float(row[25]),
                                     parse_float(row[26]), parse_float(row[27]), parse_float(row[28]),
                                     parse_float(row[29]), parse_float(row[30])])
            lounc.append(unc)
    return lounc

@typecheck
def read_usc(filename: str) -> List[USCountry]:
    """
    reads information from the specified file and returns a list of United States Greenbook country data
    """
    #return []  #stub
    # Template from HtDAP
    # loc contains the result so far
    lousc = [] # type: List[USCountry]

    with open(filename) as csvfile:

        reader = csv.reader(csvfile)
        next(reader) # skip first header line
        next(reader) # skip second header line
        next(reader) # skip third header line
        next(reader) # skip fourth header line
        next(reader) # skip fifth header line
        next(reader) # skip sixth header line
        next(reader) # skip seventh header line

        for row in reader:
            usc = USCountry(parse_int(row[0]), row[2], row[3], parse_float(row[7]))
            lousc.append(usc)
    return lousc

start_testing()

# Examples and tests for read_unc
expect(read_unc('Human Development Index (HDI) test 1.csv'), [UNC1])
expect(read_unc('Human Development Index (HDI) test 2.csv'), LOUNC1)

summary()

start_testing()

# Examples and tests for read_usc
expect(read_usc('us_foreignaid_greenbook_test1.csv'), [USCountry(1990,'Algeria','Military', 144000.00),
                                                       USCountry(1990,'Egypt','Economic', 898389000.00),
                                                       USCountry(1990,'Egypt','Economic', 900508000.00)])
expect(read_usc('us_foreignaid_greenbook_test2.csv'), [USCountry(1990,'Israel','Military', 1792260000.00),
                                                       USCountry(1990,'Israel','Military', 72774000.00),
                                                       USCountry(1990,'Jordan','Economic',3780000.00),
                                                       USCountry(1990,'Jordan','Economic',3835000.00)])

summary()
```

```
2 of 2 tests passed
2 of 2 tests passed
```

1. complile countries from greenbook

In [55]:

```
@typecheck
def countries_with_funding(lousc: List[USCountry]) -> List[str]:
    """
    takes a list of USCountry (lousc) and returns a list of country names that have recieved funding. assumes
    every name in the list has recieved funding.
    """
    # return [] # stub
    # template from list of USCountry

    # names are country names seen so far
    names = []      # type: List[str]
    for usc in lousc:
        if usc.name not in names:
            names = names + [usc.name]
    return names

# Examples and tests for countries_with_funding

start_testing()

expect(countries_with_funding([USCountry(1990,'Israel','Military', 1792260000.00),
                               USCountry(1991,'Israel','Military', 3000000.00),
                               USCountry(1991,'Israel','Economic', 47000000.00),
                               USCountry(1991,'Jordan','Economic',3780000.00),
                               USCountry(1992,'Jordan','Economic',70835000.00)]),['Israel', 'Jordan'])
expect(countries_with_funding([USCountry(1990,'United Arab Emirates','Economic',93000000.00),
                               USCountry(1991,'United Arab Emirates','Economic',71000000.00),
                               USCountry(1992,'United Arab Emirates','Military',30835000.00),
                               USCountry(1990,'Canada','Military', 90293.00),
                               USCountry(1992,'Canada','Economic', 92090.00),
                               USCountry(1993,'Canada','Economic',3780000.00),
                               USCountry(1990,'Jordan','Military',90000000.00),
                               USCountry(1991,'Jordan','Economic',82999900.00),
                               USCountry(1992,'Jordan','Military',492000000.00)]), ['United Arab Emirates',
                                                                                     'Canada', 'Jordan'])

summary()
```

```
2 of 2 tests passed
```

2. using the list of countries, filter for military aid over $50 million USD, and return the first and last years of that level of funding

In [75]:

```
@typecheck
def list_of_countries_with_years_funded(lousc: List[USCountry], loc: List[str]) -> List[List[int]]:
    """
    takes a list of USCountry (lousc) and a list of country (loc) names and returns a list of the first and
    last years of the country military funding equal to or over $50 million USD. assumes no list is empty.
    """
    # return [[]] # stub
    # template from Arbitrary-Sized and the Reference Rule

    # loy are lists of years seen so far
    loy = [] # type: List[int]
    for c in loc:
        loy = loy + [years_funded(lousc, c)]
    return loy

@typecheck
def years_funded(lousc: List[USCountry], c: str) -> List[int]:
    """
    takes a list of USCountry and a country name (c) and returns a the first and last years that that
    countries military was funded equal to or over $50 million USD. assumes no list is empty.
    """
    # return [] # stub
    # template from LOUSC with one additional parameter c

    # years are the years a given country c was funded >$50 USD seen so far
    years = []      # type: List[Optional[int]]
    for usc in lousc:
        if sum_funding(lousc, c, usc.year) >= 50000000 and usc.year not in years:
            years.append(usc.year)
    return [years[0], years[-1]]

@typecheck
```

```
@typecheck
def sum_funding(lousc: List[USCountry], c: str, y: int) -> float:
    """
    takes a list of USCountry (lousc), a country name (c), and a year (y) and sums their military funding in
    that year. assumes list contains both the country and year given
    """
    # return 0.0 # stub
    # template from list of USCountry with 2 additional parameters

    # add is the amount of money seen so far
    add = 0.0 # type: float
    for usc in lousc:
        if usc.name == c and usc.aid == "Military" and usc.year == y:
            add = add + usc.amt
    return add

start_testing()


# Examples and tests for list_of_countries_with_years_funded
expect(list_of_countries_with_years_funded([USCountry(1990,'Israel','Military', 1792260000.00),
                                            USCountry(1991,'Israel','Military', 3000000.00),
                                            USCountry(1991,'Israel','Military', 47000000.00),
                                            USCountry(1991,'Jordan','Military',3780000.00),
                                            USCountry(1992,'Jordan','Military',70835000.00)],
                                           ['Israel', 'Jordan']),[[1990, 1991], [1992, 1992]])

summary()

start_testing()

# Examples and tests for years_funded
expect(years_funded(LOUSC1, 'Algeria'), [1990, 1990])
expect(years_funded([USCountry(1990,'Israel','Military', 1792260000.00),
                        USCountry(1991,'Israel','Military', 50000000.00),
                        USCountry(1992,'Israel','Economic',3780000.00),
                        USCountry(1992,'Jordan','Economic',3835000.00)], 'Israel'), [1990, 1991])

summary()

start_testing()

# Examples and tests for sum_funding
expect(sum_funding([], "Israel", 2016), 0)
expect(sum_funding([USCountry(1990,'Israel','Military', 1792260000.00),
                        USCountry(1990,'Israel','Military', 50000000.00),
                        USCountry(1991,'Israel','Economic',3780000.00),
                        USCountry(1991,'Jordan','Economic',3835000.00)], "Israel", 1990), 1842260000.00)
expect(sum_funding([USCountry(1990,'Israel','Military', 1792260000.00),
                        USCountry(1990,'Israel','Military', 50000000.00),
                        USCountry(1991,'Israel','Economic',3780000.00),
                        USCountry(1991,'Jordan','Economic',3835000.00)], "Jordan", 1991), 0)
expect(sum_funding([USCountry(1990,'Israel','Military', 1792260000.00),
                        USCountry(1990,'Israel','Military', 50000000.00),
                        USCountry(1991,'Israel','Economic',3780000.00),
                        USCountry(1991,'Venezuela','Military',3835000.00)], "Venezuela", 1991), 3835000.0)

summary()
```

```
1 of 1 tests passed
2 of 2 tests passed
4 of 4 tests passed
```

2. using the list of countries, filter for military aid over $50 million USD, and return those countries HDIs for 1990-2018

In [ ]:

```
@typecheck
def hdis_of_funded_countries(lousc: List[USCountry], lounc: List[UNCountry],
                             loc: List[str]) -> List[List[HumanDevelopmentIndex]]:
    """
    takes a list of USCountry (lousc) and a list of country (loc) names and returns a list of the first and
    last years of the country military funding equal to or over $50 million USD. assumes no list is empty.
    """
    # return [[]] # stub
    # template from Arbitrary-Sized and the Reference Rule

    # loy are lists of years seen so far
    lohdis = [] # type: List[str]
    for c in loc:
        if [hdi_of_country(lounc, funded_countries(lousc, c))] not in lohdis:
            lohdis = lohdis + [hdi_of_country(lounc, funded_countries(lousc, c))]
            lohdis = lohdis + [hdi_of_country(lounc, funded_countries(lousc, c))]
    return lohdis


@typecheck
def hdi_of_country(lounc: List[UNCountry], c: Optional[str]) -> List[HumanDevelopmentIndex]:
    """
    takes a list of UNCountry (lounc) and a list of country names (loc) and returns a list of lists of HDIs
    of those countries
    """
    # return [[]] # stub
    # template from List[UNCountry] and 1 other arbitrary-sized parameter

    # countries are countries seen so far
    hdis = [] # type: List[HumanDevelopmentIndex]
    for unc in lounc:
        for c in loc:
            if c is None:
                hdis = hdis
            elif same_name(unc, c):
                hdis.append(convert_to_HDI(unc))
    return hdis

@typecheck
def funded_countries(lousc: List[USCountry], c: str) -> str:
    """
    takes a list of USCountry and a country name (c) and returns the name if it meets the aid value criteria.
    assumes country name c will always have a usc.name counterpart.
    """
    # return [] # stub
    # template from LOUSC with one additional parameter c

    # country is country seen so far
    country = '' # type: str
    for usc in lousc:
        while big_enough(sum_funding(lousc, c, usc.year),50000000):
            country = c
        return country

@typecheck
def same_name(unc: UNCountry, c: str) -> bool:
    """
    takes a list of UNCountry (lounc) and a country name and determines if they are the same country
    """
    # return False # stub
    # template from UNCountry

    return c == unc.name

def big_enough(a: float, b: float) -> bool:
    return a >= b

@typecheck
def convert_to_hdi(unc: UNCountry) -> List[HumanDevelopmentIndex]:
    """
    takes a UNCountry (unc) and returns its List[HumanDevelopmentIndex]
    """
    # return [] # stub
    # template from UNCountry

    return unc.HDI

start_testing()

# Examples and tests for hdis_of_funded_countries
#expect(hdis_of_funded_countries(UNC1, 'Afghanistan'), True)
#expect(hdis_of_funded_countries(UNC2, 'Afghanistan'), False)

summary()

start_testing()

# Examples and tests for hdi_of_country
#expect(hdi_of_country(UNC1, 'Afghanistan'), True)
#expect(hdi_of_country(UNC2, 'Afghanistan'), False)

summary()

start_testing()

# Examples and tests for funded_countries
expect(funded_countries([USCountry(1993,'Canada','Economic',3780000.00),
                         USCountry(1990,'Jordan','Military',90000000.00)], 'Canada'), '')
expect(funded_countries([USCountry(1991,'United Arab Emirates','Economic',71000000.00),
                         USCountry(1990,'Jordan','Military',90000000.00)],
                        'United Arab Emirates'), '')
                        'United Arab Emirates'), '')

expect(funded_countries([USCountry(1991,'United Arab Emirates','Economic',71000000.00),
                         USCountry(1991,'Jordan','Military',9000000.00)],
                        'Jordan'), 'Jordan')
expect(funded_countries([USCountry(1991,'United Arab Emirates','Military',71000000.00),
                         USCountry(1991,'Jordan','Military',9000000.00)],
                        'United Arab Emirates'), 'United Arab Emirates')

summary()

start_testing()

# Examples and tests for same_name
expect(same_name(UNC1, 'Afghanistan'), True)
expect(same_name(UNC2, 'Afghanistan'), False)

summary()

start_testing()

# Examples and tests for convert_to_hdi
expect(convert_to_hdi(UNC1), [0.298, 0.304, 0.312, 0.308, 0.303, 0.327, 0.331, 0.335, 0.339, 0.343, 0.345,
                               0.347, 0.378, 0.387, 0.4, 0.41, 0.419, 0.431, 0.436, 0.447, 0.464, 0.465, 0.479,
                               0.485, 0.488, 0.49, 0.491, 0.493, 0.496])
expect(convert_to_hdi(UNC2), [0.388, 0.395, 0.403, 0.411, 0.419, 0.427, 0.436, 0.444, 0.453, 0.462, 0.47,
                               0.479, 0.485, 0.492, 0.499, 0.506, 0.514, 0.521, 0.524, 0.535, 0.549, 0.559,
                               0.567, 0.572, 0.572, 0.588, 0.599, 0.609, 0.614])
expect(convert_to_hdi(UNC3), [None, None, None, None, None, None, None, None, None, None, 0.759, 0.767, 0.78,
                               0.82, 0.826, 0.819, 0.829, 0.829, 0.831, 0.83, 0.828, 0.827, 0.849, 0.846, 0.853,
                               0.85, 0.854, 0.852, 0.857])
expect(convert_to_hdi(UNC4), [None, None, None, None, None, None, None, None, None, None, None, None, None,
                               None, None, None, None, None, None, None, None, None, None, None, None, None,
                               None, None, None])

summary()
```

In [ ]:

```
# helper for funding_interval
@typecheck
def is_none(hdi: HumanDevelopmentIndex) -> bool:
    """
    takes an Optional[float] and returns True if it is a float. returns False if None
    """
    # return [] # stub
    # template from HumanDevelopmentIndex

    if hdi is None:
        return True
    else:
        return False

# helper for funding_interval
@typecheck
def first_encountered(lohdi: List[HumanDevelopmentIndex]) -> float:
    """
    takes a list of HDIs and returns the first non-NoneType value
    """
    # return 0.0 # stub
    # template from list of HDI with one additional parameter f

    for hdi in lohdi:
        if is_none(hdi) is False:
            return hdi

@typecheck
def list_of_funding_interval(lolohdi: List[List[HumanDevelopmentIndex]],
               loloi: List[List[int]]) -> List[List[HumanDevelopmentIndex]]:
    """
    takes a list of lists of HDIs and the years that country first and last got funding (loi)
    and returns the HDIs of the years that country had funding. if the country is still getting funding, it will
    return the HDI of 2018
    """
    # hdis are lists of the first and last the hdis seen so far
    hdis = [] # type: List[List[HumanDevelopmentIndex]]
    for lohdi in lolohdi:
        for loi in loloi:
            hdis.append(funding_interval(lohdi, loi))
    return hdis

# helper for list_of_funding_interval
@typecheck
def funding_interval(lohdi: List[HumanDevelopmentIndex], loi: List[int]) -> List[HumanDevelopmentIndex]:
    """
    takes a list of HDIs and the years that country first and last got funding (loi)
    and returns the HDIs of the years that country had funding. if the country is still getting funding, it will
    return the HDI of 2018.
    """
    # return [] # stub
    # ... (lohdi, f, l) # template

    f = loi[0]
    l = loi[-1]

    if is_none(lohdi[f-1990]) and is_none(lohdi[l-1990]):
        return []
    elif is_none(lohdi[f-1990]) and is_none(lohdi[l-1990]) is False:
        return [first_encountered(lohdi), lohdi[l-1990]]
    else:
        return [lohdi[f-1990], lohdi[l-1990]]

# helper for funding_to_percentage
@typecheck
def percent_change(lohdi: List[HumanDevelopmentIndex]) -> float:
    """
    takes two non-NoneType numbers in a list and returns the percent change between the first to the second
    """
    # return 0.0 # stub
    # ...(lohdi) # template
    return ((lohdi[1] - lohdi[0]) / lohdi[0]) * 100

@typecheck
def categorize(lof: List[float]) -> List[int]:
    """
    takes a list of floats and categorizes them into >2%, <2%, <-2%, >-2%, and no significant change
    """
    # return [] # stub
    # template from Arbitrary-Sized

    # below are all accumulators of different categories of accumulators
    over_2 = 0 # type: int
    under_2 = 0 # type: int
    under_neg_2 = 0 # type: int
    over_neg_2 = 0 # type: int
    no_significant_change = 0

    for f in lof:
        if f >= 2:
            over_2 = over_2 + 1
        elif 2 > f > 0.5:
            under_2 = under_2 + 1
        elif f <= -2:
            under_neg_2 = under_neg_2 + 1
        elif -2 < f < -0.5:
            over_neg_2 = over_neg_2 + 1
        else:
            no_significant_change = no_significant_change + 1

    return [over_2, under_2, under_neg_2, over_neg_2, no_significant_change]

@typecheck
def fractions(lof: List[float]) -> List[float]:
    """
    takes a list of floats returns them in fractions of categories
    """
    # return [] # stub
    # ... (lof) # template

    cat = categorize(lof)
    total = len(lof)

    po2 = (cat[0] / total) * 100
    pu2 = (cat[1] / total) * 100
    pun2 = (cat[2] / total) * 100
    pon2 = (cat[3] / total) * 100
    pnc = (cat[4] / total) * 100

    return [po2, pu2, pun2, pon2, pnc]

@typecheck
def funding_to_percentage(lolohdi: List[List[HumanDevelopmentIndex]], lousc: List[USCountry]) -> List[float]:
    """
    combines a lot of these functions to go from a list of funding to a list of percentages
    """
    # return [] #stub
    # template from Arbitrary-Sized and the Reference Rule
    lop = [] # type: List[float]

    for lohdi in lolohdi:
        lop = lop + [percent_change
                     (list_of_funding_interval
                      (lohdi, list_of_countries_with_years_funded(lousc, countries_with_funding(lousc))))]
    return lop

def combine_all(lounc: List[UNCountry], lousc: List[USCountry]) -> List[float]:
    """
    takes a list of float percentages and returns their weight in fractions
    """
    return fractions(funding_to_percentage
                     (filter_for_country
                      (lounc, list_of_countries_with_years_funded(lousc, countries_with_funding(lousc))), lousc))

start_testing()

# Examples and tests for is_none
expect(is_none(None), True)
expect(is_none(0.321), False)

summary()

start_testing()

# Examples and tests for first_encountered
expect(first_encountered([None, None, None, None, None, None, None, None, None, None, 0.759, 0.767, 0.78,
                         0.82, 0.826, 0.819, 0.829, 0.829, 0.831, 0.83, 0.828, 0.827, 0.849, 0.846, 0.853,
                         0.85, 0.854, 0.852, 0.857]), 0.759)
expect(first_encountered([None, None, None, None, None, None, None, None, None, None, None, 0.767, 0.78,
                         0.82, 0.826, 0.819, 0.829, 0.829, 0.831, 0.83, 0.828, 0.827, 0.849, 0.846, 0.853,
                         0.85, 0.854, 0.852, 0.857]), 0.767)

summary()

start_testing()

# Examples and tests for funding_interval
expect(funding_interval([0.298, 0.304, 0.312, 0.308, 0.303, 0.327, 0.331, 0.335, 0.339, 0.343, 0.345,
                        0.347, 0.378, 0.387,0.4, 0.41, 0.419, 0.431, 0.436, 0.447, 0.464, 0.465, 0.479,
                        0.485, 0.488, 0.49, 0.491, 0.493, 0.496], [1992, 1998]),
                        [0.312, 0.339])
expect(funding_interval([0.388, 0.395, 0.403, 0.411, 0.419, 0.427, 0.436, 0.444, 0.453, 0.462, 0.47,
                        0.479, 0.485, 0.492, 0.499, 0.506, 0.514, 0.521, 0.524, 0.535, 0.549, 0.559,
                        0.567, 0.572, 0.572, 0.588, 0.599, 0.609, 0.614], [1992, 1998]), [0.403, 0.453])
expect(funding_interval([None, None, None, None, None, None, None, None, None, None, 0.759, 0.767, 0.78,
                            0.82, 0.826, 0.819, 0.829, 0.829, 0.831, 0.83, 0.828, 0.827, 0.849, 0.846, 0.853,
                            0.85, 0.854, 0.852, 0.857], [1998, 2002]), [0.759, 0.78])
expect(funding_interval([None, None, None, None, None, None, None, None, None, None, None, None, None,
                               None, None, None, None, None, None, None, None, None, None, None, None,
                               None, None, None], [2006, 2009]), [])

summary()


start_testing()

# Examples and tests for percent_change
expect(percent_change([0.783, 0.923]), 17.879948914)
expect(percent_change([0.923, 0.783]), -15.167930661)

summary()

start_testing()

# Examples and tests for categorize
expect(categorize([-2, -9, 14.3, 12.4, 0.2, 0.1, 1.3]), [2, 1, 2, 0, 2])
expect(categorize([-1.2, -9.3, -0.2, -0.1, 0.2, 0.1, 9, 9.3, 14.0, 12.3, 1.3]), [4, 1, 1, 1, 4])

summary()

start_testing()

# Examples and tests for fractions
expect(fractions([-2, -9, 14.3, 12.4, 0.2, 0.1, 1.3]), [28.57142857142857, 14.285714285714285, 28.57142857142857,
                                                        0.0, 28.57142857142857])
expect(fractions([-1.2, -9.3, -0.2, -0.1, 0.2, 0.1, 9, 9.3, 14.0, 12.3, 1.3]), [36.36363636363637,
                                                                                9.090909090909092,
                                                                                9.090909090909092,
                                                                                9.090909090909092,
                                                                                36.36363636363637])

summary()
```

```
start_testing()

# Examples and tests for funding_to_percentage

expect(funding_to_percentage([[0.321, 0.356, 0.369], [0.321, 0.356, 0.369], [None, 0.728, 0.889],
                              [None, None, 0.921]],
                            [USCountry(1990,'Israel','Military', 1792260000.00),
                            USCountry(1991,'Israel','Military', 50000000.00),
                            USCountry(1992,'Israel','Military',3780000.00),
                            USCountry(1990,'Canada','Military', 90293.00),
                            USCountry(1992,'Canada','Military', 92090.00),
                            USCountry(1993,'Canada','Military',3780000.00),
                            USCountry(1990,'Jordan','Military',90000000.00),
                            USCountry(1991,'Jordan','Military',82999900.00),
                            USCountry(1992,'Jordan','Military',492000000.00),
                            USCountry(1990,'United Arab Emirates','Military',93000000.00),
                            USCountry(1991,'United Arab Emirates','Military',71000000.00),
                            USCountry(1992,'United Arab Emirates','Military',30835000.00)]),
                             [14.953, 0, 22.115, 0])

summary()

start_testing()

# Examples and tests for all_together(lousc: List[USCountry], lounc: List[UNCountry]) -> List[float]:
expect(percent_change([0.783, 0.923]), 17.879948914)
expect(percent_change([0.923, 0.783]), -15.167930661)

summary()
```

Final Graph/Chart

Now that everything is working, you **must** call `main` on the intended information source in order to display the final graph/chart:

In [ ]:

```
main('Human Development Index (HDI).csv')
```

In [ ]:

```
# Remove the # sign from the beginning of the line with your instructor's name. For example, if you are
# in Jessica's class, remove the # from  the line that says INSTRUCTOR_NAME = 'Jessica'. If you are in
# in Meghan's class, remove the # from the line that says INSTRUCTOR_NAME = 'Meghan'.

# RUN THIS CELL (press the "Run" button) AFTER YOU REMOVE THE # SIGN

# INSTRUCTOR_NAME = 'Jessica'
# INSTRUCTOR_NAME = 'Meghan'
```

In [ ]:

```
# Be sure to select ALL THE FILES YOU NEED (including csv's)
# when you submit. As usual, you cannot edit this cell.
# Instead, run this cell to start the submission process.
from cs103 import submit

COURSE = 0
ASSIGNMENT = 0

if(INSTRUCTOR_NAME == 'Jessica'):
    COURSE = 50466
    ASSIGNMENT = 517964 # Final submission
elif(INSTRUCTOR_NAME == 'Meghan'):
    COURSE = 48359
    ASSIGNMENT = 518017 # Final submission

submit(COURSE, ASSIGNMENT)

# If your submission fails, check to see that you have specified a value for the INSTRUCTOR_NAME variable in the
# cell above AND have run that cell. If you  have done so and your submission is still failing, SUBMIT ANYWAY
# by downloading your files and uploading them to Canvas. You can learn how on the page
# "How to submit your Jupyter notebook" on our Canvas site.
```

In [ ]: