Program Planning and Structuring

As mundane as this seems, this is critical for the vast majority of problems (basically every problem whose solution isn't super trivial such as a program that only requires a couple lines of code). You do this passively with most problems you face anyways:

- Identifying the problem
- Isolating important pieces of information
- Planning on how to approach the problem
- Laying out the framework of your solution
- Implementing your solution

This set of notes will be doing a rather in-depth guide of all five of these explanations. To better highlight these steps and why topics/concepts such as comments and testing are important, I will be showing how I approach an exam practice problem and explaining in detail every step I do. Do be aware that I am going very in-depth, but in practice, the time you take to do all these steps will vary (coding quizzes may be 15-20 minutes while labs can be done over a whole week for example).

Additionally, I will also show unit testing (testing your code for certain cases). This is similar to the team module 5 lab (the writing test cases lab) but on a much smaller scale thankfully. Do note that on a quiz or exam, you can't test your code in an IDE nor should you be adding test code to your answer: you will have to test by hand if you have the time.

The prompt

The prompt is based on last fall's "2022 Exam 1 Old Questions and Practice Problem Set" (you can click the link to go to the PDF). Specifically, this is Q9. The prompt is as follows:

> ```
> Write a program that will ask a user to enter names and ages of people, stopping when an age
> of 0 is entered (and not processing that person). The program should collect this information,
> and then output the average age, the name of the oldest person, and the name of the youngest
> person. Assume no two people have the same age.
> ```

For the sake of simplicity, I will assume the user will be nice. In other words, I will get ages that are integers as this reflects real life (your age in real life is typically an integer). On a real problem though, never assume! Usually the prompt states if you need to deal with invalid inputs or you can assume the user will be nice. If you do not know for sure, ask a PT or your prof for clarification.

Identifying the problem + isolating imporant info

These two generally go hand-in-hand. As you read the prompt, you should be picking up on keywords and phrases that hint at certain structures and reveal the input and output. As I read the prompt, I see the following:

1. "ask a user to enter names and ages of people"
2. "stopping when an age of 0 is entered"
3. "collect this information"
4. "output the **average** age... name of **oldest** person, name of **youngest person**
5. "Assume no two people have the same age"

Now you might be wondering, that's basically the whole prompt. That's also my point: more than likely, the prompt given to you consists mainly of important information, which is why it is imperative to pay special attention to the prompt.

With these five pieces of information, I now have an idea of how the program will be formatted and what I need in order to complete it:

- (1) + (2) implies using a `while` loop with `input()`:
  - (1) tells me that I need to use `input()` as the user is asked to enter entries
  - (1) doesn't specify how many names or ages but (2) specifies the condition in which to *stop*, so this is a `while` loop (as `while` loops run as long a condition is met)
- (3) tells me that I need to store this information. With a *non-constant* number of entries, I may use a list
- (4) tells me three extremely important details:
  - "average age": the average is defined as the sum of all the numbers divided by the quantity of numbers (may need `sum()` and `len()` to add up the values in the list and get the size of the list respectively)
  - "name... oldest person": I need two variables, one to keep track of the name and the other to keep track of the age
  - "name... youngest person": I need two more variables, one to keep track of the name and the other to keep track of the age
- (5) simplifies the process for (4): as no two people have the same age, I do not need to worry about tie-breakers or outputting multiple names

I haven't even planned the actual structure, and I already have all this information. This is why planning (and comments) are important, which you'll see me implement in the next section: I can keep track of all this information and not have to risk forgetting an important detail by gambling on whether or not I remember everything instead of writing this down.

Planning on how to approach the problem

This planning will be done as code (more specifically comments). For these notes, I will be writing down my entire thought process: note that this is not necessary especially if time is a constraint. It may be best to directly write down your comments immediately than to write out your thoughts as psuedocode and then do the comments and code structuring. This is also dependent on your comfort and personal assessment of your skill: you should do what makes you feel most confident in regards to finishing the assignment on time and creating the most accurate code to the maximum of your extent.

Anyways, getting back on track, at this point, I want to create a mental sketch of the program *flow*. In other words, I want to see the entire process from getting the input to showing the output. As pseudocode, it looks like this:

```
set names and ages list
set two input() (name and age)

while the age is not zero
    store the name and age in their respective lists
    ask again for input

calculate the average
find the youngest and oldest person

output average
output name of oldest person
output name of youngest person
```

Now this is just a sketch: note that I did not exactly show how I wanted to calculate the average nor how I would find the oldest and youngest person. But the inputs should be simple enough, so I'll leave that to the side. At this point, I'm thinking about my calculations. For the average, it's simple enough: `average = sum(ages) / len(ages)`. This is by definition of average (sum of numbers over size of the set of numbers).

However, I run into my first major problem: how do I correctly find the oldest and youngest person? If I try to sort the list of ages, how do I now know which name goes with what age? Initially, the nth age is matched with the nth name since they are inputted into the list at the same time, and thus should be at the same location. If I sort a list, this connection is no longer true since the list of ages had its values shuffled. I now think of two solutions:

1. Use a dictionary
2. Find a different way to calculate average and find the youngest and oldest people

(1) is viable but could be rather complex. This code would now require me to correctly use a list, three list functions (`sum()` , `len()` , and `.sort()` / `sorted()`), a dictionary, and a while loop. I would like to do something simpler to reduce the complexity of my program, which has two big benefits:

- Easier to grade (I've made mistakes when grading before but in my experience, the simpler/more straight forward the code is, the more accurate and quicker I can grade it)
- Easier to code and debug

So let's go with (2). But how? Note that all I need for average is the sum and the quantity: it does not matter what the actual numbers are. Also, for the youngest and oldest people, I don't care about the people in the middle: I just need to make sure I have the youngest and oldest people. What if I just updated the youngest and oldest as I processed the inputs?

If you're confused at what I'm getting at, let me show you an example: let's say I needed you to find the smallest number in a list of numbers. There are two ways:

- List all the numbers then sort them
- As you go through the numbers, keep a note of what is the current smallest number and update this as you go down the list

I want to do the second option. By doing this, I eliminate the need to store the actual values, which means I do not need a list nor do I need a dictionary. However, I need more variables to keep track of everything. Let's update the psuedocode:

```
set variables for sum and size (the quantity of numbers) to 0 (for calculating average)
set variables for name and age of oldest person
set variables for name and age of youngest person

set two input() for current name and age
while age is not 0
    add the age to sum
    add one to length

    if this age is smaller than current youngest age
        update youngest person info
    if this age is larger than current oldest age
        update oldest person info

    ask for input again
```

```
calculate average (sum / length)

output average, name of oldest person, name of youngest person
```

Well, I actually have another problem. I need to compare the person's info with the youngest and oldest people. But who's initially the youngest and oldest people? The first person I process. How do I know that I am processing the first person? By setting a flag (a `True/False` flag to be specific, so a boolean basically).

You may be asking at this point why I didn't set default values for the name and age instead? I'm not really a big fan of default values personally. I made the assumption that the ages are always numeric. But what if the prompt specified that I could get *any input* for the names and ages? There is always the chance that the input matches my default value exactly. Of course, there are ways to figure out a default value that even when overlapped, it wouldn't affect the code (such as setting the age to a letter), but that's more complexity than needed.

If I set some random flag variable to `True` then change it to `False` after processing one person, then this works for all scenarios since the flag value is entirely based on whether I process a person, not on default values. I won't claim that it's best practice over default values (sometimes you will use default values), but for this program, I just prefer to use a flag as it's simpler. The new psuedocode becomes:

```
set variables for sum and size (the quantity of numbers) to 0 (for calculating average)
set variables for name and age of oldest person
set variables for name and age of youngest person

set two input() for current name and age
set a flag that represents if I have processed a person or not
while age is not 0
    add the age to sum
    add one to length

    if this age is smaller than current youngest age or the flag is False
        update youngest person info
    if this age is larger than current oldest age or the flag is False
        update oldest person info

    set the flag to True
    ask for input again

calculate average (sum / size)

output average, name of oldest person, name of youngest person
```

You can check if the flag is `False` then change to `True`, but it doesn't really matter. The key thing here is that the flag needs to be changed to something other than `False` after I process the first person and set that person's info to be the youngest and oldest person so I have a person of reference to compare with for future people I process. Repeatedly setting the flag to `True` is fine since the flag is no longer reset back to `False`.

Due to my assumption that the ages will be integers, I do not need extra code to deal with improper inputs. There's no further checks I need to do, so I can start with making my comments.

If you believe this psuedocode has problems still, you would be correct! I set my sum and size to be 0 at the start. But my average is the sum divided by the size. What if the very first input, I get an age of 0? The while loop would be skipped, and my average would be 0/0: an error! I need one last check when calculating the average: if the size is 0, just set the average to "N/A". As this case is not explicitly discussed in the prompt, I get to decide what to do. The actually final psuedocode is as follows:

```
set variables for sum and size (the quantity of numbers) to 0 (for calculating average)
set variables for name and age of oldest person
set variables for name and age of youngest person

set two input() for current name and age
set a flag that represents if I have processed a person or not
while age is not 0
    add the age to sum
    add one to length

    if this age is smaller than current youngest age or the flag is False
        update youngest person info
    if this age is larger than current oldest age or the flag is False
        update oldest person info

    set the flag to True
    ask for input again

calculate average (sum / size if size is not 0, otherwise, set to N/A)

output average, name of oldest person, name of youngest person
```

Laying out the framework

I do this purely with comments. This will set up the skeleton/framework of my program and will help guide my coding.

Comments do not explain your code, they just summarize your code. It's like back in high school where you annotate each paragraph with a quick bullet point/short summary of what it's about. The same thing applies with comments and code. Obviously, add as much detail now as you need to, but for the sake of time and code readability, more concise comments are preferred.

It's hard to really describe what sort of comments I pesronally like to write, so I'll just write them out, and you can get a feel for what I mean. The way to write comments is more or less a stylistic preference but in general, it is neater to write concise comments that still provide sufficient information. You don't have to copy my style, I just want you to at least have a grasp of understanding why we use comments and more or less how to use them in regards to code planning. Referring to the most recent psuedocode, I will now set up my comments:

```
# Set average variables (total + size)

# Set youngest and oldest people variables (name + age for each)

# Get user input for name and age

# Set flag for is processing first person

# Process and continue getting input while age is not 0

    # Update total and size

    # Update youngest person if person is younger or is first person

    # Update oldest person if person is older or is first person

    # Update flag, get input again

# Calculate the average

# Output average, name of youngest, name of oldest
```

Like I said earlier, this is just an example of how I may write this. I think this is enough personally, or maybe I would condense some comments together. But in a class where comments that show your thought process can get you partial credit, too much detail is better than not enough. Note that I even indented some comments: this is me anticipating that these sections of code are in the `while` loop I know that I need. If you couldn't think that far ahead or it's just confusing, feel free to properly indent your comments (for clarity reasons, it won't crash Python if you don't indent comments) later.

Implementing the solution

With the framework down, let's get started with the coding. I will go through each of these sections one by one then show all the code together at the end:

- Initial variables + setup
- The while loop
- Output

Initial Variables and Setup

I consider the first four comments to be classified under this:

```
# Set average variables (total + size)

# Set youngest and oldest people variables (name + age for each)

# Get user input for name and age

# Set flag for is first person
```

Using the comments as reference, I can now make my code:

```
# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0

# Get user input for name and age
current_name = input("Enter a name: ")
```

```
current_age = int(input("Enter their age: "))

# Set flag for processing first person
has_processed_first_person = False
```

Nothing fancy here. Note that `total` and `size` must be 0 because they are the only logical default values: if you haven't processed any ages, your current sum is 0 and the quantity of ages you have processed (the `size`) is also 0. Also, do not use `sum` as your variable name because it's actually a built in functon (`sum()`)! It's tempting, but it's not recommended whatsoever and also why I said "total" not "sum" in my comment.

Furthermore, don't forget that the age is a number, so I casted the `input()` for `age` to `int` with `int()`. The last thing to note is my choice of variable names. They may seem long, and there are shorter variable names available, but they are descriptive enough to be unique and instantly recognizeable of what value they store and their purpose. `youngest_name` is clearly the name of the youngest person. However, a variable name like `x1` is very vague (in this context).

The while loop

I consider the next five comments (comments 5-10) to be the part of the `while` loop. These comments are:

```
# Process and continue getting input while age is not 0

    # Update total and size

    # Update youngest person if person is younger or is first person

    # Update oldest person if person is older or is first person

    # Update flag, get input again
```

Using these comments as a template, I will write the following code:

```
# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))
```

The condition was that the `while` loop runs while the age is not 0, in other words, while `current_age != 0`. I then update `total` and `size` appropriately.

I have two `if` statements instead of an `if` and `elif`. This is because these conditions are independent of each other: just because the person is the youngest doesn't mean they can't be the oldest too! Although this sounds weird, this can happen! Note that I said that we have to set the youngest and oldest people to the very first person we process, so the future people we process can be compared against an actual person. If I used an `elif`, I would fail to update the oldest person as technically I could input a negative integer. Of course, if the ages were limited to just non-negative (not positive because 0 is a valid input) integers, then this wouldn't be an issue, but my assumption is just integers, not non-negative integers.

After going through the two `if` statements, I update the flag to `True` and ask for input again (do not forget to cast the `current_age` to int).

Output

The two last comment (which are also the only remaining comments without code written under it) is for the output. The comment is the following:

```
# Calculate average

# Output average, name of youngest, name of oldest
```

Usually there are example outputs so you are aware of exactly what gets printed out and in what format, but since there are no example outputs, I'll just make sure to output the results with a label that makes it clear which value is which:

```
# Calculate average
average = "N/A"
```

```
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

By default, I assume that the average is -1 (this is the case if `size` is 0). The other case is if `size` is not 0 (hence my `if` statement condition is `size > 0`), and I need to calculate the average and update `average` with the result. Note that `size != 0` is also valid, but it just makes more sense to me in the moment to use `size > 0` since I know that either my size is 0 or it's a positive number since I only change it by incrementing it.

The `print` statements are pretty straight forward. I just use f-strings to help put in the values into their respective strings. No formatting is specified, so this should be fine. My final code (along with a header I added so you can find the full code easier if you ever come back to these notes in the future):

In [ ]:
```
"""
2022 Exam 1 Old Exam Questions and Practice Problems: Question 9

Made by PT Anthony Pham
Last updated: 20 9 2023
"""

# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0

# Get user input for name and age
current_name = input("Enter a name: ")
current_age = int(input("Enter their age: "))

# Set flag for processing first person
has_processed_first_person = False

# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))

# Calculate average
average = "N/A"
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

Testing

Well, let's go to testing the code. Thankfully, the minimum test cases is not fifty or remotely that high. But the question becomes, what do I need to test? For these sorts of programs, we want to test if each part of our code is done correctly. More precisely, I want to ensure the following works:

1. Input is done properly

2. Updating the youngest person is done properly
3. Updating the oldest person is done properly
4. Calculating the average is done properly

With this in mind, let's think of some scenarios (your typical, edge, and corner) for each of these. For (1), any test case will do as if the input is not done properly, I'll just get an incorrect output. So, my focus is with (2), (3), and (4).

For (2), we have five scenarios I need to consider:

1. The people are inserted in ascending age order
2. The people are inserted in descending age order
3. The people are inserted in random order (neither the first or last person is the youngest)
4. Only one person is inputted
5. Multiple people share the same name

For sorting questions, I need to think of the different ways inputs can be put in. Sometimes they are sorted (ascending/descending order) or they are not. Also, I need to test if the setting the youngest and oldest person to the first person works, so the fourth scenario is included. The prompt said that all the ages are unique but not the names, so this is why statement five exists.

For (3), we have five scenarios I need to consider:

1. The people are inserted in ascending age order
2. The people are inserted in descending age order
3. The people are inserted in random order (neither the first or last person is the youngest)
4. Only one person is inputted
5. Multiple people share the same name

These five follows the same logic as the logic I wrote in the paragraph for (2).

For (4), we have two scenarios I need to consider:

1. `size` is 0
2. `size` is non-zero

Either I output "N/A" for a size of zero, or I output a proper average, so I just need two scenarios (one for each).

So currently, we have twelve test cases (one for each scenario)! However, all five of the scenarios in (2) and (3) can be combined as both can be checked simultaneously. Also, the 2nd scenario for (4) is met for all test cases we make for testing (2) and (3). This means that we only have six test cases that will verify all ten scenarios (named as "statement_number-scenario_number" so for example, the 2nd scenario for (4) would be 4-2):

- Three people in ascending age order (verifies 2-1, 3-1, 4-2)
- Three people in descending age order (verifies 2-2, 3-2, 4-2)
- Four people in "random order" (verifies 2-3, 3-3, 4-2)
- Only one person (verifies 2-4, 3-4, 4-2)
- Multiple people share the same name (verifies 2-5, 3-5, 4-2)
- No person (verifies 4-1)

I will opt to use the following as my test cases (each test case has the inputs in order of when I will input them):

- Tony 1 Luna 2 Gabe 3 Sophie 0
- Tony 3 Luna 2 Gabe 1 Sophie 0
- Tony 3 Luna 1 Gabe 4 Sophie 2 Luke 0
- Tony 1 Luna 0
- Tony 2 Luna 1 Luna 3 Sophie 0
- Tony 0

The important part of this program is dependent on the ages which is why I kept the names the same and in the same order (except for the fifth test case where I deliberately duplicated a name to test if the program works for multiple people with the same names). I also chose very nice numbers so the average calculation is simple to do by hand.

My expected outputs are as follows (shown in the order of average, youngest name, oldest name):

- 2.0 Tony Gabe
- 2.0 Gabe Tony
- 2.5 Luna Gabe
- 1.0 Tony Tony
- 2.0 Luna Luna
- N/A

As the default values the names were set to are "", I should get no names for the very last test case. I put a large header at the top of

each test case so that you can more easily identify which test case is which.

In [ ]:
```
"""
Test Case 1:
Tony 1 Luna 2 Gabe 3 Sophie 0

Expected Output:
The average of the ages is: 2.0
The name of the youngest person is: Tony
The name of the oldest person is: Gabe
"""

# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0

# Get user input for name and age
current_name = input("Enter a name: ")
current_age = int(input("Enter their age: "))

# Set flag for processing first person
has_processed_first_person = False

# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))

# Calculate average
average = "N/A"
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

```
The average of the ages is: 2.0
The name of the youngest person is: Tony
The name of the oldest person is: Gabe
```

In [ ]:
```
"""
Test Case 2:
Tony 3 Luna 2 Gabe 1 Sophie 0

Expected Output:
The average of the ages is: 2.0
The name of the youngest person is: Gabe
The name of the oldest person is: Tony
"""

# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0
```

```
# Get user input for name and age
current_name = input("Enter a name: ")
current_age = int(input("Enter their age: "))

# Set flag for processing first person
has_processed_first_person = False

# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))

# Calculate average
average = "N/A"
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

```
The average of the ages is: 2.0
The name of the youngest person is: Gabe
The name of the oldest person is: Tony
```

In [ ]:

```
"""
Test Case 3:
Tony 3 Luna 1 Gabe 4 Sophie 2 Luke 0

Expected Output:
The average of the ages is: 2.5
The name of the youngest person is: Luna
The name of the oldest person is: Gabe
"""

# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0

# Get user input for name and age
current_name = input("Enter a name: ")
current_age = int(input("Enter their age: "))

# Set flag for processing first person
has_processed_first_person = False

# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
```

```
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))

# Calculate average
average = "N/A"
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

```
The average of the ages is: 2.5
The name of the youngest person is: Luna
The name of the oldest person is: Gabe
```

In [ ]:
```
"""
Test Case 4:
Tony 1 Luna 0

Expected Output:
The average of the ages is: 1.0
The name of the youngest person is: Tony
The name of the oldest person is: Tony
"""

# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0

# Get user input for name and age
current_name = input("Enter a name: ")
current_age = int(input("Enter their age: "))

# Set flag for processing first person
has_processed_first_person = False

# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))

# Calculate average
average = "N/A"
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

```
The average of the ages is: 1.0
The name of the youngest person is: Tony
The name of the oldest person is: Tony
```

In [ ]:
```
"""
```

```
Test Case 5:
Tony 2 Luna 1 Luna 3 Sophie 0

Expected Output:
The average of the ages is: 2.0
The name of the youngest person is: Luna
The name of the oldest person is: Luna
"""

# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0

# Get user input for name and age
current_name = input("Enter a name: ")
current_age = int(input("Enter their age: "))

# Set flag for processing first person
has_processed_first_person = False

# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))

# Calculate average
average = "N/A"
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

```
The average of the ages is: 2.0
The name of the youngest person is: Luna
The name of the oldest person is: Luna
```

In [ ]:

```
"""
Test Case 6:
Tony 0

Expected Output:
The average of the ages is: N/A
The name of the youngest person is:
The name of the oldest person is:
"""

# Set average variables (total + size)
total = 0
size = 0

# Set youngest and oldest people variables (name + age for each)
youngest_name = ""
youngest_age = 0
oldest_name = ""
oldest_age = 0

# Get user input for name and age
current_name = input("Enter a name: ")
current_age = int(input("Enter their age: "))
```

```
# Set flag for processing first person
has_processed_first_person = False

# Process and continue getting input while age is not 0
while current_age != 0:
    # Update total and size
    total += current_age
    size += 1

    # Update youngest person if person is younger or is first person
    if current_age < youngest_age or not has_processed_first_person:
        youngest_name = current_name
        youngest_age = current_age

    # Update oldest person if person is older or is first person
    if current_age > oldest_age or not has_processed_first_person:
        oldest_name = current_name
        oldest_age = current_age

    # Update flag, get input again
    has_processed_first_person = True
    current_name = input("Enter a name: ")
    current_age = int(input("Enter their age: "))

# Calculate average
average = "N/A"
if size > 0:
    average = total / size

# Output average, name of youngest, name of oldest
print(f"The average of the ages is: {average}")
print(f"The name of the youngest person is: {youngest_name}")
print(f"The name of the oldest person is: {oldest_name}")
```

```
The average of the ages is: N/A
The name of the youngest person is:
The name of the oldest person is:
```

Final Remarks

And there you go, I have finished my code and have verified its functionality with six distinct test cases. As you can see, this entire process from start to finish can be quite lengthy. Me writing everything out and showing each test case being proved individually certainly made this process look even longer so do keep that in mind. In reality, the process won't be this long, but the steps will still be more or less the same.

Hopefully, this document has helped you understand why you are urged to use comments so much and how program planning can be very useful for your assignments and future projects.