UNDERSTANDING AND TRACING CODE

REMINDERS

PREVIOUSLY:

1. Understand the Problem
2. Write your Signature and Purpose
3. Identify Examples/Test Cases
4. Solve your Examples by Hand
5. Identify Your Data Types
6. Write your Outline
7. Fill in the Gaps
8. Code!
9. Test

PREVIOUSLY:

1. Understand the Problem
2. Write your Signature and Purpose
3. Identify Examples/Test Cases
4. Solve your Examples by Hand
5. Identify Your Data Types
6. Write your Outline
7. Fill in the Gaps
8. Code!
9. Test <-- What do we do when something goes wrong??

WHAT IS DEBUGGING?

Identifying where your **mental model** of how the code works doesn't align with the actual reality of how the code works

WHAT IS DEBUGGING?

Identifying where your **mental model** of how the code works doesn't align with the actual reality of how the code works

mental model: the concept you have in your brain for how something works

EXAMPLE

Mental model: "I wrote a function that checks if each number in a given list is greater than the previous number"

```python
def check_increasing(nums : list[int]) -> bool:
    for i in range(len(nums)-1):
        if nums[i+1] > nums[i]:
            return True
    return False
```

EXAMPLE

Mental model: "I wrote a function that checks if each number in a given list is greater than the previous number"

```
def check_increasing(nums : list[int]) -> bool:
    for i in range(len(nums)-1):
        if nums[i+1] > nums[i]:
            return True
    return False
```

Maybe at first glance this seems true, but let's check some unit tests...

```python
# checks if each number in a list is greater than the previous number
def check_increasing(nums : list[int]) -> bool:
    for i in range(len(nums)-1):
        if nums[i+1] > nums[i]:
            return True
    return False
```

INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT
[1, 2, 3]	True	True
[4, 1, 2, 3]	False	False
[1, 10, 2, 11]	False	True

What do these unit tests tell us?

OUR UNIT TESTS TELL US

- It is definitely not working correctly
- It is marking some things as False so it's not always saying true
- The function is NOT actually checking if each number in a list is greater than the previous number
    - **[L1]** So what do you think it might be doing instead?

INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT
[1, 2, 3]	True	True
[4, 1, 2, 3]	False	False
[1, 10, 2, 11]	False	True

INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT
[1, 2, 3]	True	True
[4, 1, 2, 3]	False	False
[1, 10, 2, 11]	False	True

Maybe it's not checking the whole list? Let me investigate the parts of the code that stop execution (returns)

```python
def check_increasing(nums : list[int]) -> bool:
    for i in range(len(nums)-1):
        if nums[i+1] > nums[i]:
            return True # <-- !!! Hang on, this happens immediately!
    return False
```

In this case, it's only checking if the *second item in the list is greater than the first*

Oops!

WHAT HAPPENS WHEN OUR CODE IS COMPLEX?

"this code takes a list of fahrenheit temperatures as a command line argument and finds their median in celsius"

```
def to_c_median(temps):
    for i in range(len(temps)):
        t = temps[i]
        temps[i] = (t-32)*5/9
    temps = sorted(temps)
    middle = len(temps) // 2
    return temps[middle]
```

```
def main(temps):
    tmps = []
    for t in temps:
        tmps.append(int(t))
    print(to_c_median(tmps))

if __name__ == "__main__":
    temps = sys.argv[1].split()
    main(temps)
```

PAUSE: ON IF NAME == MAIN

What is this??

```
if __name__ == "__main__":
    temps = sys.argv[1].split()
    main(temps)
```

This is some fancy special syntax that just means "when this file is run using python <name_of_file> then run this stuff. Otherwise skip it."

This becomes necessary as we **import** our code into other files so we can use our functions elsewhere (like in unit tests!)

Usually does any small things you want for testing and then calls `main()`

UNPAUSE: UNIT TESTS

"this code takes a list of fahrenheit temperatures as a command line argument and finds their median in celsius"

INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT
[32, 212, 122]	50	50
[32, 32, 14, 212]	0	0
[32, 14, 212, 122]	25	0

Oh no I'm not sure what's going wrong??

IN THIS CASE WE HAVE TO DO SOME DIGGING

Ideally, we would have our code broken into functions so we can pinpoint what step is breaking. But uhoh! This is just one long function!

```
def to_c_median(temps):
    for i in range(len(temps)):
        t = temps[i]
        temps[i] = (t-32)*5/9
    temps = sorted(temps)
    middle = len(temps) // 2
    return temps[middle]
```

At that point it becomes helpful to trace our code!

TRACING BY HAND

```
def to_c_median(temps):
    for i in range(len(temps)):
        t = temps[i]
        temps[i] = (t-32)*5/9
    temps = sorted(temps)
    middle = len(temps) // 2
    return temps[middle]
```

TRACING WITH PRINT

LET'S MAKE A SMALL MODIFICATION TO OUR MAIN

```
def main(temps):
    tmps = []
    for t in temps:
        tmps.append(int(t))
    print("full median:", to_c_median(tmps))
    print("oh I forgot what the median was:", to_c_median(tmps))
```

Our mental model: "this code with convert the list of °F temperatures to °C and then find the median, which will be printed twice"

WHAT ACTUALLY HAPPENS?

INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT
[32, 212, 122]	50 ... 50	50... 10
[32, 32, 14, 212]	0 ... 0	0 ... -14.77

Hand Tracing Across Functions

```
def main(temps):
    tmps = []
    for t in temps:
        tmps.append(int(t))
    print("full median:", to_c_median(tmps))
    print("oh I forgot what the median was:", to_c_median(tmps))

def to_c_median(temps):
    for i in range(len(temps)):
        t = temps[i]
        temps[i] = (t-32)*5/9
    temps = sorted(temps)
    if len(temps) % 2 == 0:
        return (temps[len(temps)//2] + temps[(len(temps)//2) + 1]) / 2
    return temps[len(temps) // 2]
```

WHY DOES THIS HAPPEN?

In python, lists are **mutable**.

That means that if we change the value of the list, even in a separate function, that change will be propogated back even across a function call.

[L2] What other types do we know that are mutable?

YOUR TURN! DRAW OUT THE TRACE [C1]

"This program takes a dict of puchased items. If they're worth more than $50 then shipping is free. And then also applies a 20% discount to the items"

```
def apply_discount(prices):
    for item in prices:
        prices[item] *= 0.80
    return prices

def compute_ship(prices):
    cost = sum(prices.values())
    shipping = 10
    if cost >= 50.0:
        shipping = 0
    return shipping
```

```
def main():
    prices = {
    "shirt": 30.0, "pants": 30.0
    }
    disc = apply_discount(prices)
    disc_s = sum(disc.values())
    ship = compute_ship(prices)
    print("Total:", disc_s + ship)
```

CHALLENGE PROBLEM

"I wrote a function that removes every odd number in a given list"

```
def remove_odds(nums_list: list[int]):
    for num in nums_list:
        if num % 2 == 1:
            nums_list.remove(num)
    return nums_list
```