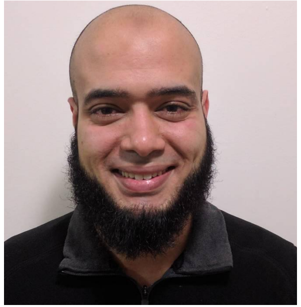
Data Structures

Effective Coding and Debugging for linked list

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Draining your time

- Linked list challenges
  - Often involve deletion, insertion, and **relinking** of multiple nodes
- Linked list coding may drain your time
  - Several coding bugs
  - Several run-time errors
  - Hard to visualize
- Thinking time
  - It will also take time to get the right idea, but this is the ok part
- Take this video very seriously
  - It provides coding and debugging tips to save you as much time as possible!

Before thinking/coding

- List several test cases and their answers
- Consider lists of different sizes / even & odd size
- Consider the problem - and what might be tricky test cases
- This will make you aware of what kind of issues you need to handle
- Later, evaluate your code against all test cases
- Common mistakes
    - Avoiding the above before thinking about a tentative solution
    - To test over weak cases
    - We can all miss some good case, but the code should at least work for several cases

Thinking skills

- KISS: Keep it simple stupid
  - Some problems seem impossible at first - but you can do them! Calm down
  - Most problems can be coded elegantly in a few lines.
    - Sadly, too many instructors and websites take a longer path
  - Try to make things as simple as possible
- Sketch out your thoughts on paper, NOT on your computer
  - A very common mistake is to rush to code your idea
  - This can waste hours with run-time errors and buggy outputs
  - Think deeply when writing down your ideas. **Draw each step**.
    - Use addresses if needed
  - Verify & trace different test cases
  - This will boost your abstract thinking skills, save your time and boost your confidence
- Finally: Compare with my codes. Learn from them!

Coding skills

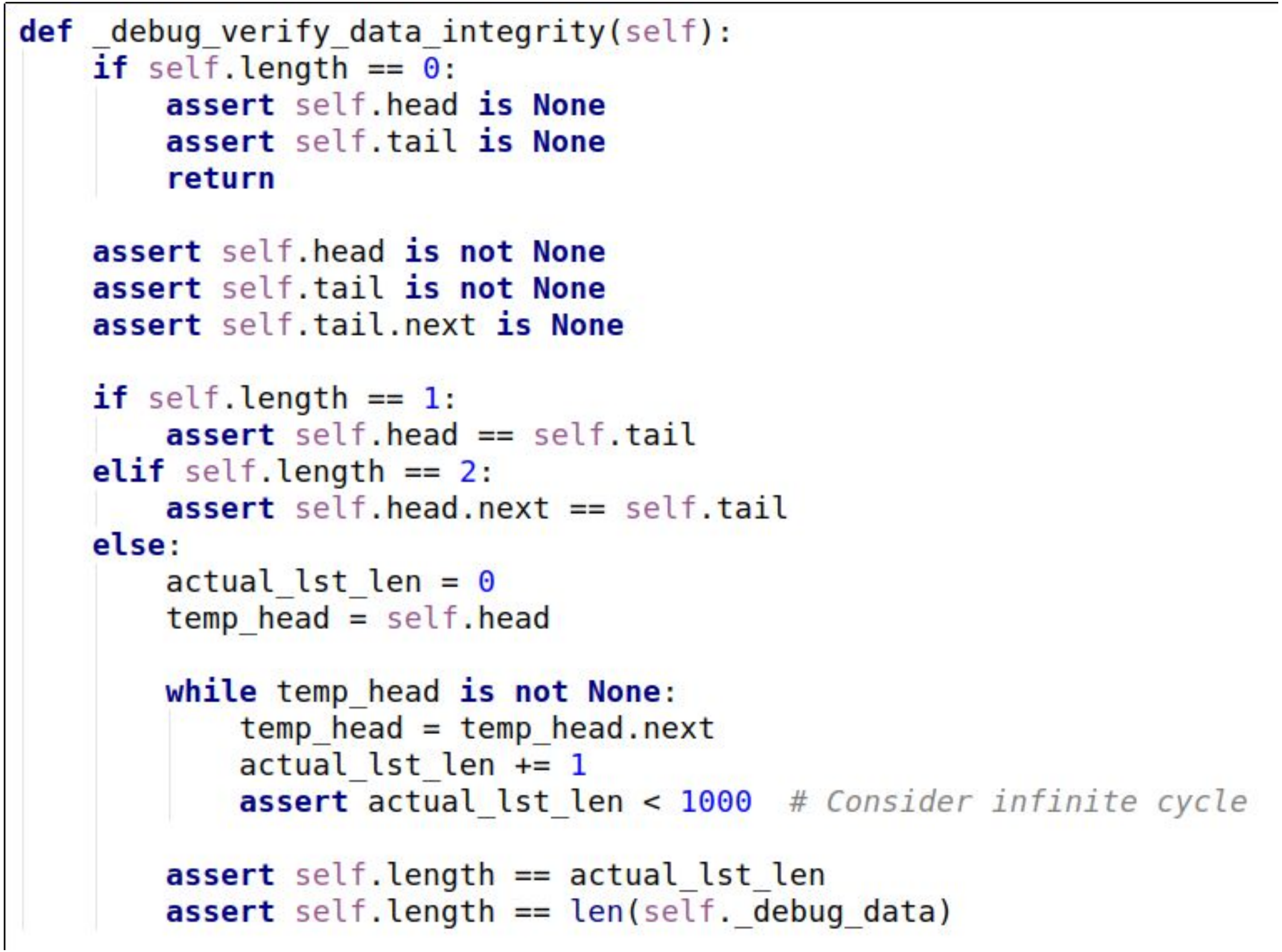
- Think modular
  - Avoid writing a lengthy function (even with code comments)
  - Every time you notice something can be converted to a helpful method, do it
  - Pick very clear names
  - The more you solve, the more you will notice how functions can be reused or recycled!
  - Document your functions: input, output and any conditions
- Tip: before coding decide the minimum number of needed elements
  - 1 node? 2 nodes? 3 nodes?
  - Without enough nodes, some approaches are destined to fail

Coding Mistakes

- Incorrect algorithm
- Wrong order of operations
  - Sometimes you need to take a copy of a node’s next before cancelling its next
- Run time errors
  - node.next will throw an RTE if node is None
  - node.next.next will throw an RTE if node.next is None
  - Double-check them every time you use them in your code. Ensure:
    - The logic is correct
    - You are verifying against null first

Data Integrity

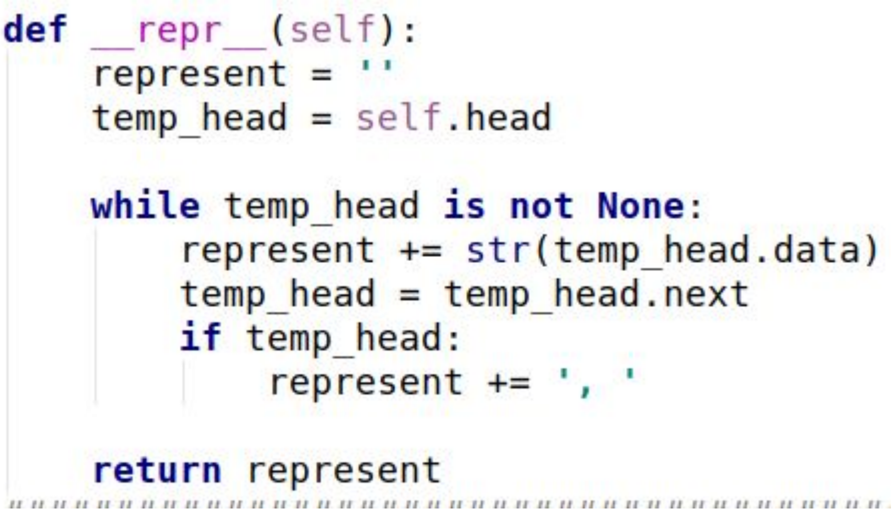
- Data integrity is the overall accuracy, completeness, and **consistency** of data
- What is our data?
  - head, tail and length
    - E.g. Head and tail should be null if empty
    - E.g. Length must be really the length of the items
- Write a function that verifies a linked list is correct
- Run it after your main algorithm is done (or intermediate if possible)
  - It can catch so many mistakes!



```python
def _debug_verify_data_integrity(self):
    if self.length == 0:
        assert self.head is None
        assert self.tail is None
        return

    assert self.head is not None
    assert self.tail is not None
    assert self.tail.next is None

    if self.length == 1:
        assert self.head == self.tail
    elif self.length == 2:
        assert self.head.next == self.tail
    else:
        actual_lst_len = 0
        temp_head = self.head

        while temp_head is not None:
            temp_head = temp_head.next
            actual_lst_len += 1
            assert actual_lst_len < 1000  # Consider infinite cycle

        assert self.length == actual_lst_len
        assert self.length == len(self._debug_data)
```

String for comparisons

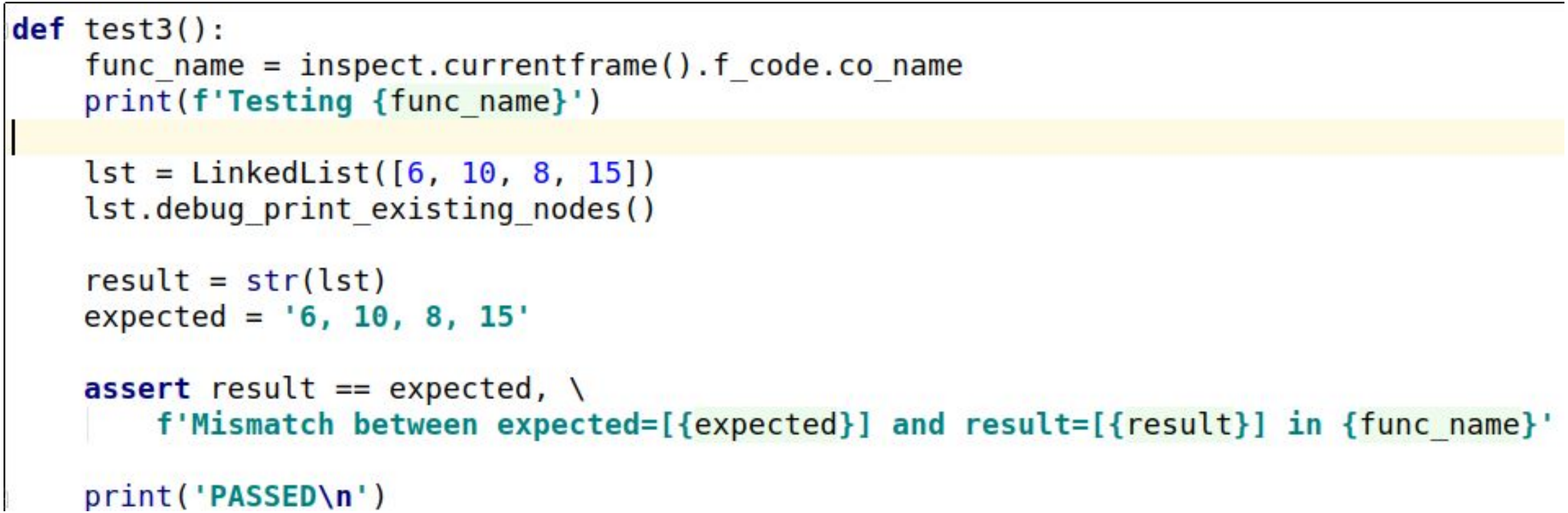
- To easily compare your function result vs expected output, let’s convert the data into a string, using this function:



```
def __repr__(self):
    represent = ''
    temp_head = self.head

    while temp_head is not None:
        represent += str(temp_head.data)
        temp_head = temp_head.next
        if temp_head:
            represent += ', '

    return represent
```

Testing

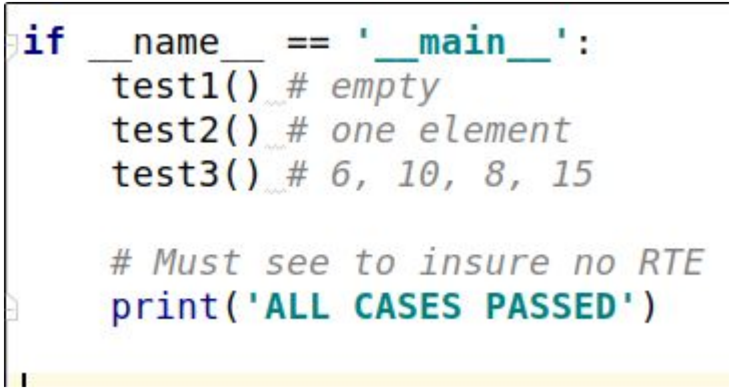
- For each test case, develop its list & operations
    - Compare its content with the expected output
- From main, run all of your test functions



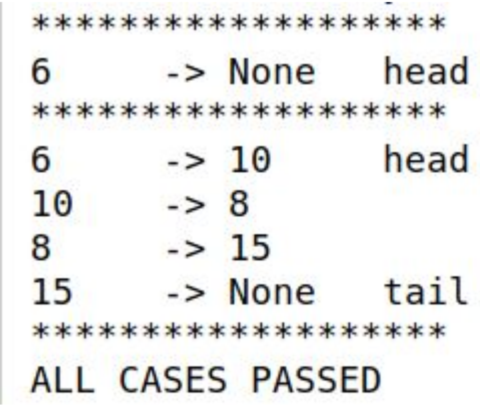
```python
def test3():
    func_name = inspect.currentframe().f_code.co_name
    print(f'Testing {func_name}')

    lst = LinkedList([6, 10, 8, 15])
    lst.debug_print_existing_nodes()

    result = str(lst)
    expected = '6, 10, 8, 15'

    assert result == expected, \
        f'Mismatch between expected=[{expected}] and result=[{result}] in {func_name}'

    print('PASSED\n')
```

Testing

- In main, run all test cases. Last msg must be printed to verify no RTE
  - debug_print_existing_nodes() visualize the linked list for you
  - debug_print_address() will print nodes and their addresses
    - Useful to verify addresses



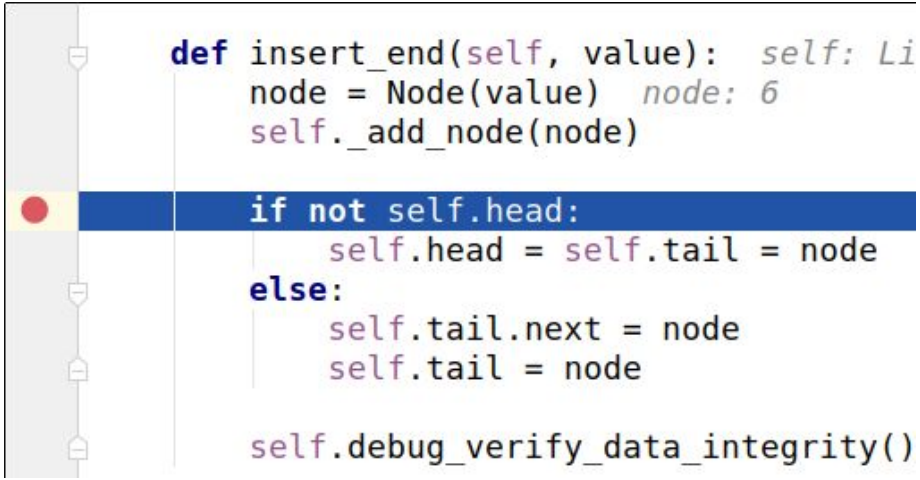
```
if __name__ == '__main__':
    test1() # empty
    test2() # one element
    test3() # 6, 10, 8, 15

    # Must see to insure no RTE
    print('ALL CASES PASSED')
```



```
*******************
6       -> None    head
*******************
6       -> 10      head
10      -> 8
8       -> 15
15      -> None    tail
*******************
ALL CASES PASSED
```

Debugger

- Your IDE’s debugger will also help you discover several mistakes!
- Make sure you're comfortable using your IDE's debugger
- Prepare test cases. **Draw** out the steps
- Run the debugger and confirm it matches!
- Verify data integrity



```
def insert_end(self, value):  self: Li
    node = Node(value)  node: 6
    self._add_node(node)

    if not self.head:
        self.head = self.tail = node
    else:
        self.tail.next = node
        self.tail = node

    self.debug_verify_data_integrity()
```

Code Template

- Base your algorithms on my code template
- Utilize the debugging facilities in it

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”