Teaching *Computation* not just *Code*

Two tools 'most everyone uses,
two tools Dave uses too,
and one tool nobody should use for CS1

Notional Machines and teaching about *Computation*

Students (and teachers) sometimes focus on *code* and *output*

Rich and accurate (or, at least, extensible) understanding of computation matters!

- "Notional Machine" [DuBoulay '81]
  - should be actively taught, not implicit
- Alternative notiontal machines [Krishnamurthi, others]
  - "Dictionary notional machine"
    - computation = update "state" (program counter + variables and objects) to get result
  - "Substitution notional machine"
    - computation = rewrite (code + input) until you get the result
- Visualizers for notional machines [Sorva, others]
  - Should be "presenters" or "realizers" or "evincers", use vision, sound, touch, as needed

Visualizing the Dictionary Notional Machine

There are a rich variety of visualization tools for the D.N.M.

Standard Debugger

- Shows position in code, stack of variables, associated values
- Scales well to professional level
- Obscures subtlety of state

Python Tutor [Guo '21]

- Limited amount of code, number of steps
- Highlights structure of state

Demos, as time permits!

Challenges of the Dictionary Notional Machine

```python
def copyElement(dest, src, x, y):
    """
    Copy src[x][y] over dest[x][y]

    >>> a1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    >>> a10 = [[10, 20, 30], [40, 50, 60], [70, 80, 90]]
    >>> copyElement(a1, a10, 2, 0)
    >>> a1  # note the 70 below
    [[1, 2, 3], [4, 5, 6], [70, 8, 9]]
    """
    checkTotal = (sum(sum(row) for row in src) +
                  sum(sum(row) for row in dest))  # total value
    increase = src[x][y] - dest[x][y]  # e.g. changing 3 to 7 is increase 4
    dest[x][y] = src[x][y]
    newTotal = (sum(sum(row) for row in src) +
                sum(sum(row) for row in dest))
    assert checkTotal + increase == newTotal
```

Does this code work?

Challenges of the Dictionary Notional Machine

```
def copyElement(dest, src, x, y):  dest: [[1, 3, 5], [2, 4, 6], [3, 5, 7
    checkTotal = (sum(sum(row) for row in src) +  checkTotal: 99
                  sum(sum(row) for row in dest))  # total value
    increase = src[x][y] - dest[x][y]  # e.g. changing 3 to 7 is increas
    dest[x][y] = src[x][y]
    newTotal = (sum(sum(row) for row in src) +
                sum(sum(row) for row in dest))
    assert checkTotal + increase == newTotal

def demo():
    (magic, byTwo) = setup()
    copyElement(byTwo, magic, 2, 2)
```

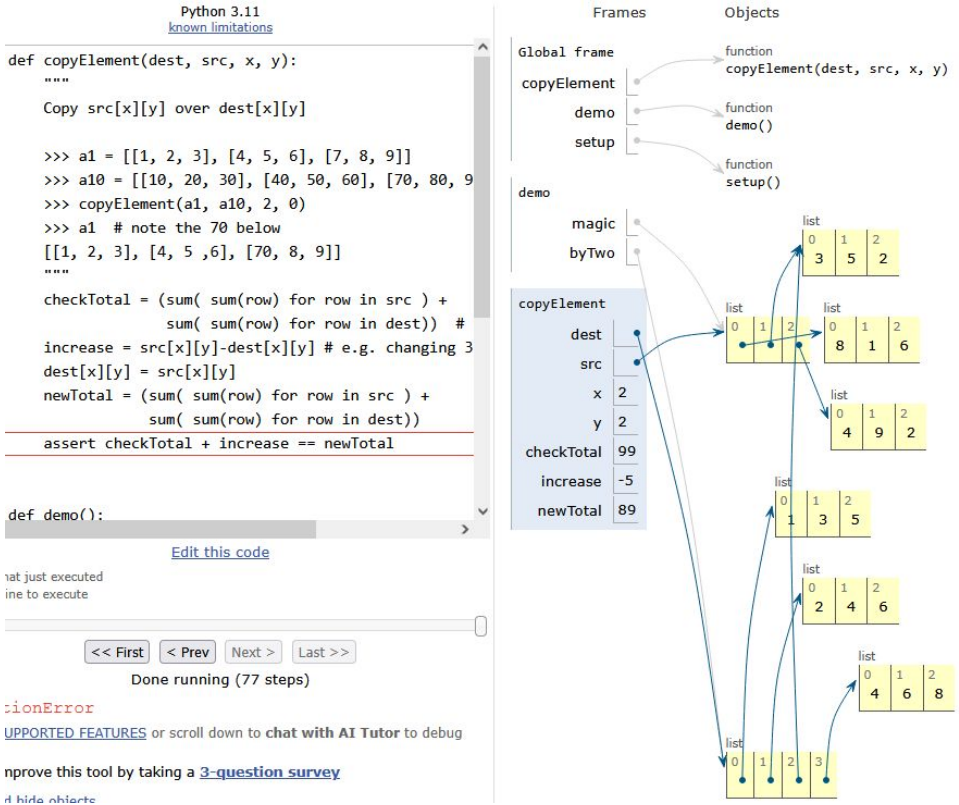
copyElement()

Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)

- checkTotal = {int} 99
- dest = {list: 4} [[1, 3, 5], [2, 4, 6], [3, 5, 7], [4, 6, 8]]
- src = {list: 3} [[8, 1, 6], [3, 5, 7], [4, 9, 2]]
- x = {int} 2
- y = {int} 2

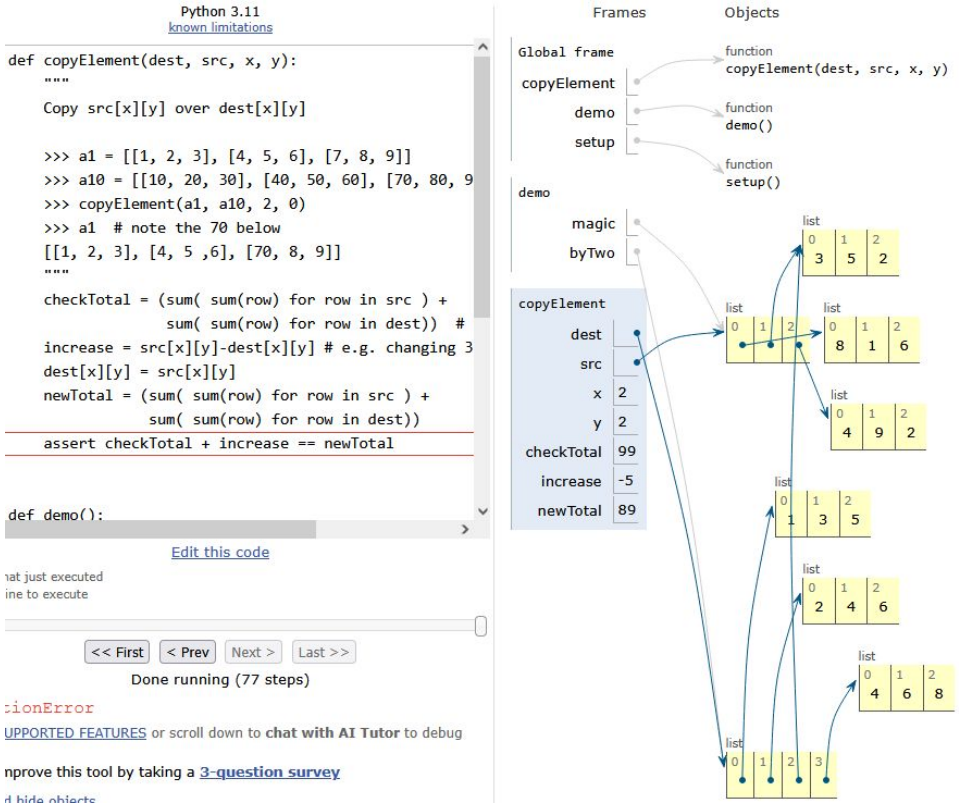
Does this tool help us understand why the code does not work?

Challenges of the Dictionary Notional Machine



What about this tool?

Challenges of the Dictionary Notional Machine



What about this tool?

For imperative code,
We must know ***both***
(name x scope) ⟶ object
and
object ⟶ value
relationships.

Visualizer must present both!

Visualizing the Substitution Notional Machine

Fewer tools; at undergraduate level, only suitable for pure-functional code

"Refactor" menu in IDE

- helps convince students that substitution is "real" and industry-relevant
- many steps not available, e.g., refactor `if False:` or `6*7`

N-Dolphin [Wonnacott, Reichard '23]

- abstract, symbolic, re-orderable substitution machine
- core pure-functional language, currently renders as Python
- substitution, plus a bridge to proofs/discrete math [Wonnacott & Osera]

DrRacket Stepper (specific to racket language) [Clements, Flatt, Felleisen '01]

Demos, as time permits!

Challenges of the Substitution Notional Machine

No (normal) loops, so much think about everything in terms of recursion

N-Dolphin 0.12.0, April 2025 davew@yao

```
def fib(n: int) -> int:
  if (n<3):
    return 1
  else:
    return (fib((n-1))+fib((n-2)))

fib(5)
```

Challenges of the Substitution Notional Machine

No (normal) loops, so must think about everything in terms of recursion

N-Dolphin 0.12.0, April 2025 davew@yao

```
def fib(n: int) -> int:
  if (n<3):
    return 1
  else:
    return (fib((n-1))+fib((n-2)))

(1
  if (5<3) else
  (fib((5-1))+fib((5-2))))
```

N-Dolphin 0.12.0, April 2025 davew@yao

```
def fib(n: int) -> int:
  if (n<3):
    return 1
  else:
    return (fib((n-1))+fib((n-2)))

(1
  if False  else
  (fib((5-1))+fib((5-2))))
```

```
(fib((5-1))+fib((5-2)))
```

```
(3+2)
```

```
5
```

Updating tools

All tools should communicate in ways that reach the user

Audio output is vital for visually-impaired programmers/students

- Still a work in progress
- Particularly weak for state information of the D.N.M. and imperative code

Choice of language should be allowed

- Python Tutor supports many languages, though not Chapel :-(
- N-Dolphin could be adapted to support Chapel (I think) relatively easily
    - Just need to map A.S.T. to printed output

"This is extremely nasty"

Why Dave hates teaching with Jupyter (or other) "notebook"s

As time permits, demo/rant about "spring surprise"

The "Notebook" notional machine … not quite the DNM

Spring Surprise.ipynb

File Edit View Insert Runtime Tools Help

ımands + Code + Text Run all

Share Gemini

RAM Disk

```
[33] legsPerSpider = 8
```

```
[34] nSpiders = 1
```

```
print("Total spider legs:", nSpiders*legsPerSpider)
```

```
Total spider legs: 8
```

Note that we can intermix *explaination* text with *code*, and include mathematics in our explanations if we want to.

```
[36] nLadyBugs = 20
```

```
[37] legsPerInsect = 8
```

```
[38] print("Total lady bug legs:", nLadyBugs * legsPerInsect)
```

```
Total lady bug legs: 160
```

If we had more time, I could show code that produces a nice **graph** or something, to make clear why the computational scientists really like this tool.

Consider this notebook

What happens when we change nSpiders?

What about when we correct legsPerInsect?