APS106

UNIVERSITY OF TORONTO

Lists: indexing and slicing.

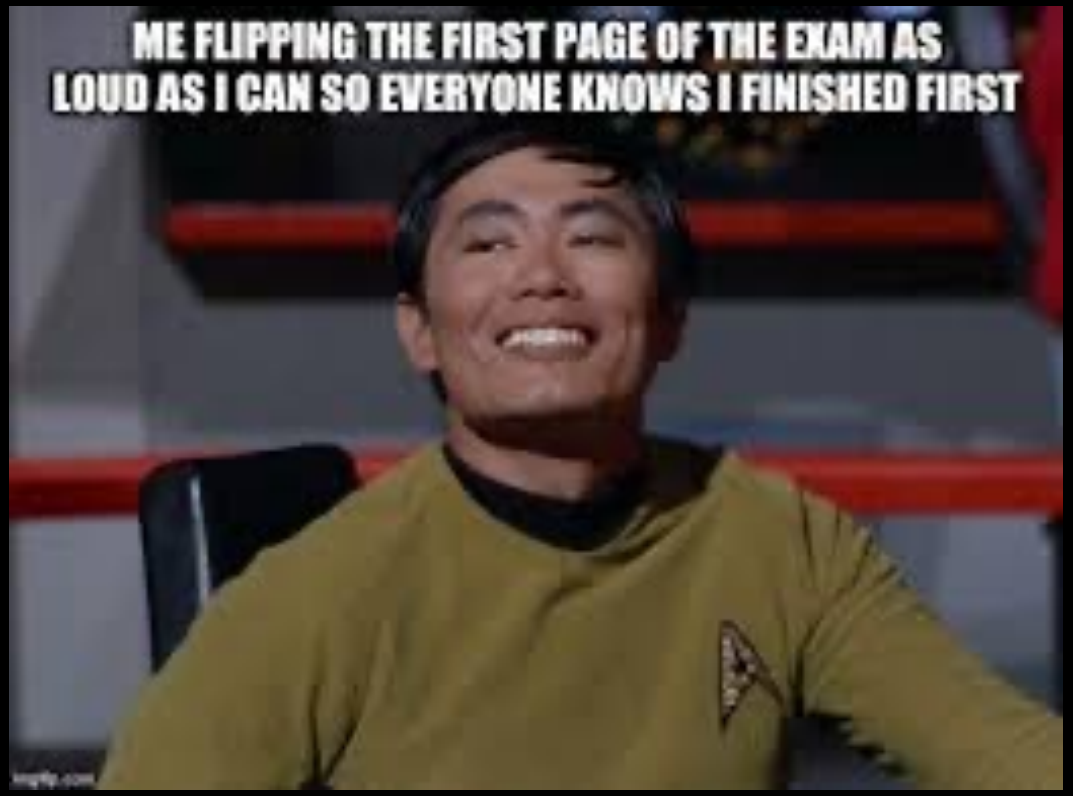
Week 7 | Lecture 1 (7.1)

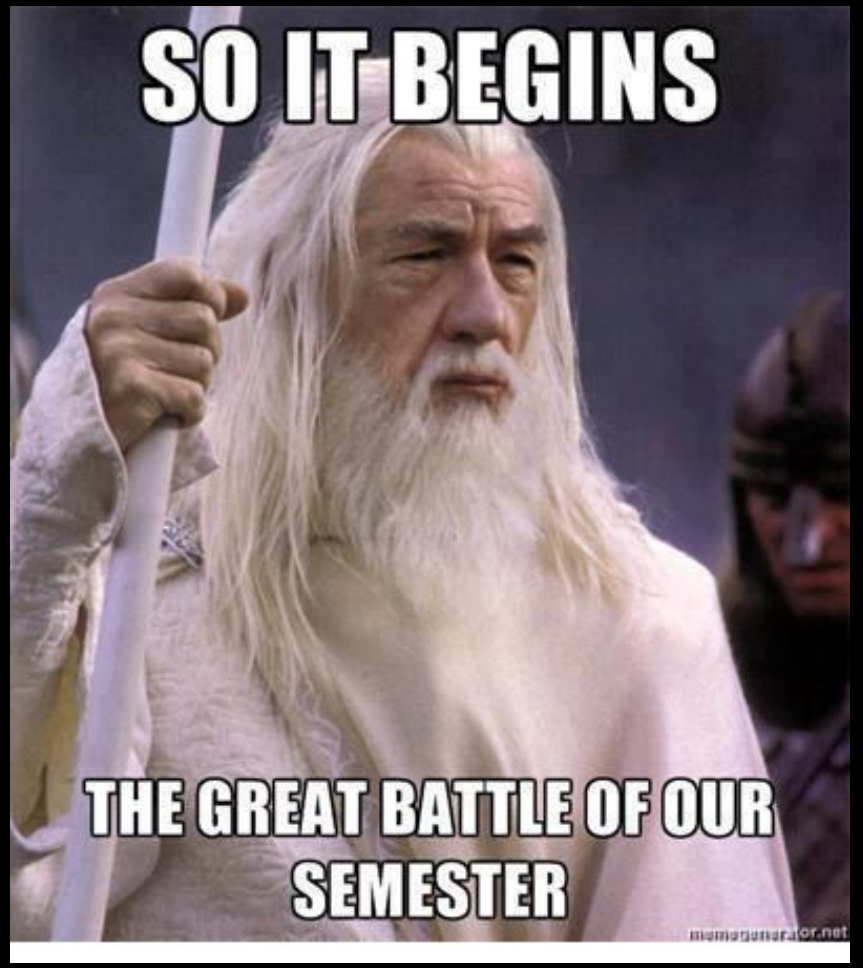
if nothing else, write #cleancode

This Week's Content

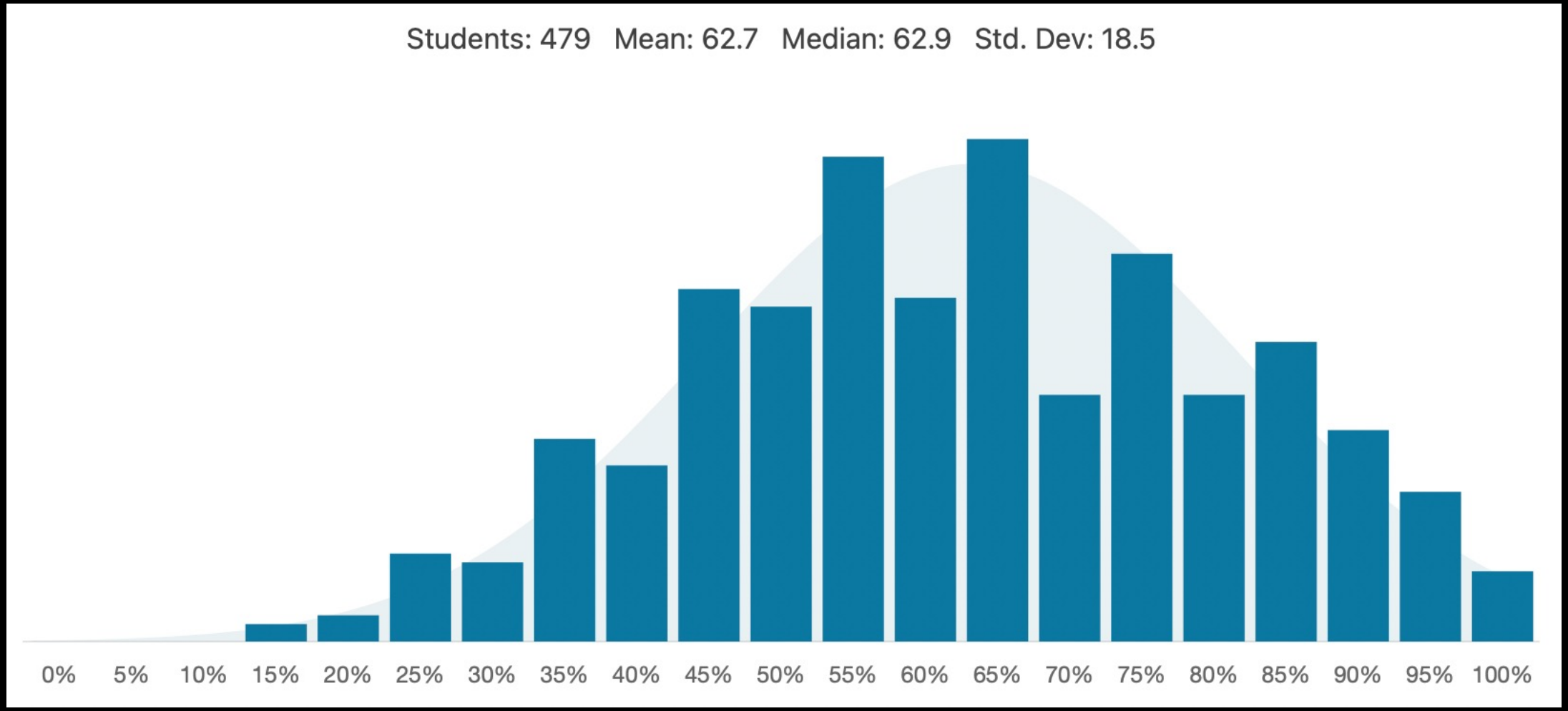
- **Lecture 7.1**
  - **Lists: indexing and slicing**
- **Lecture 7.2**
  - Lists: nested lists and looping
- **Lecture 7.3**
  - Design Problem! Cryptography…

But First! The Midterm...

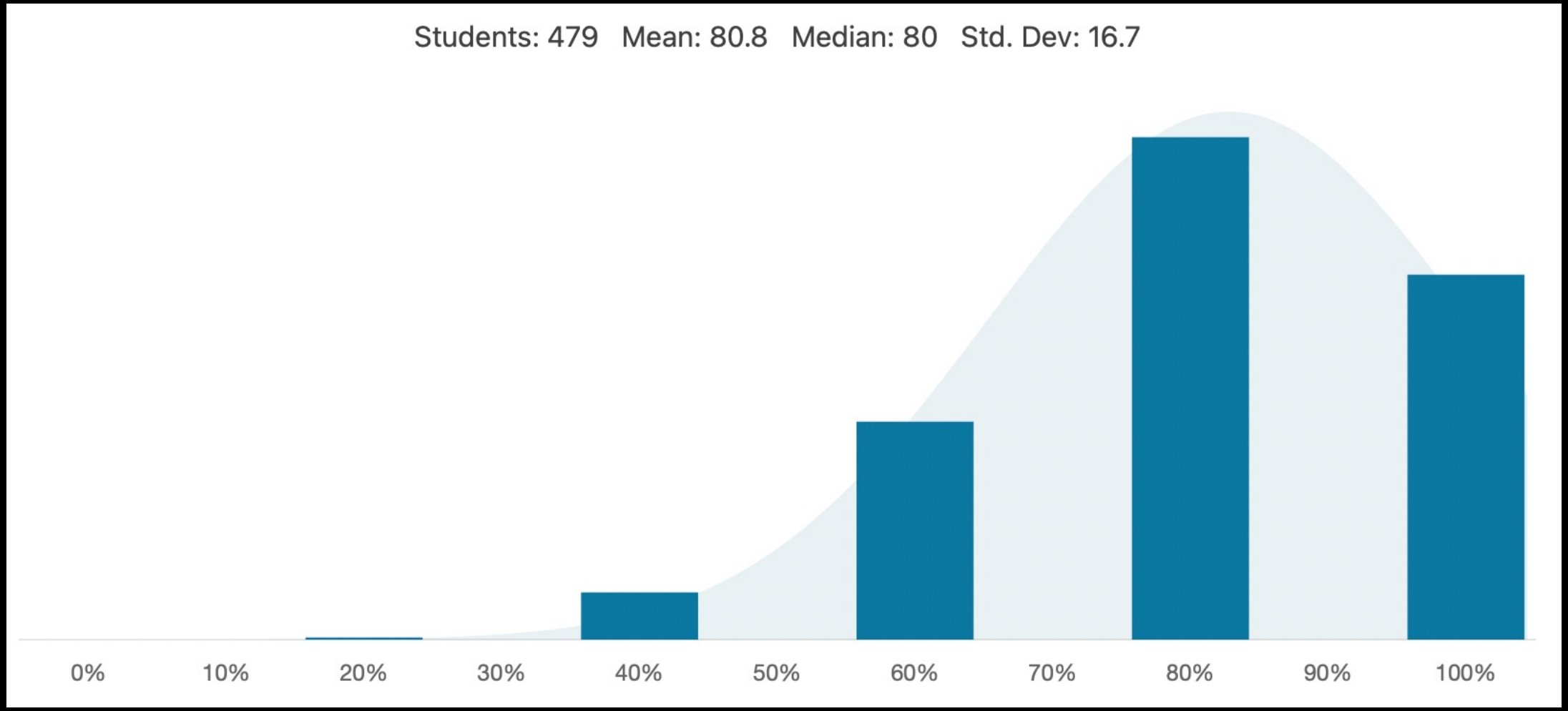




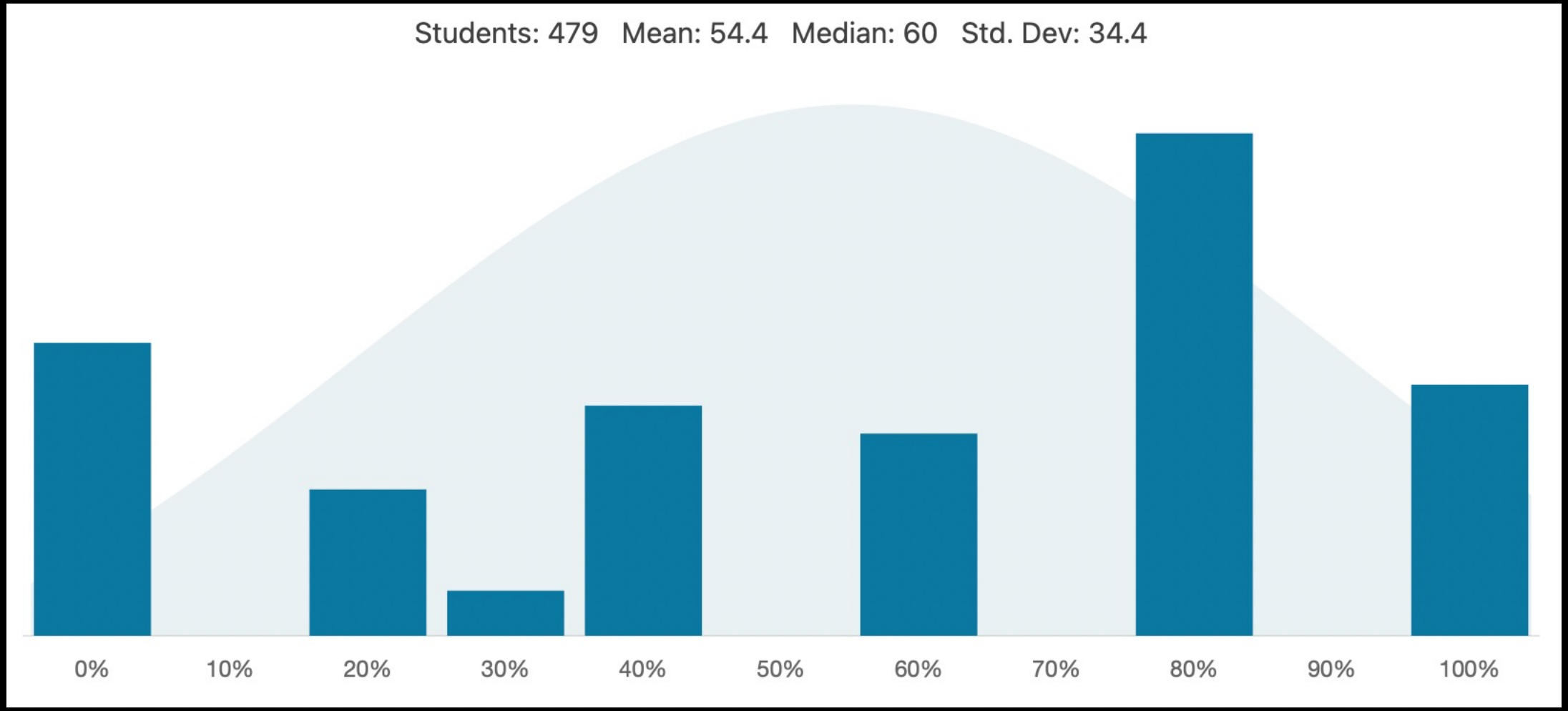
Midterm 1


Students: 479 Mean: 62.7 Median: 62.9 Std. Dev: 18.5
0%
5%
10%
15%
20%
25%
30%
35%
40%
45%
50%
55%
60%
65%
70%
75%
80%
85%
90%
95%
100%

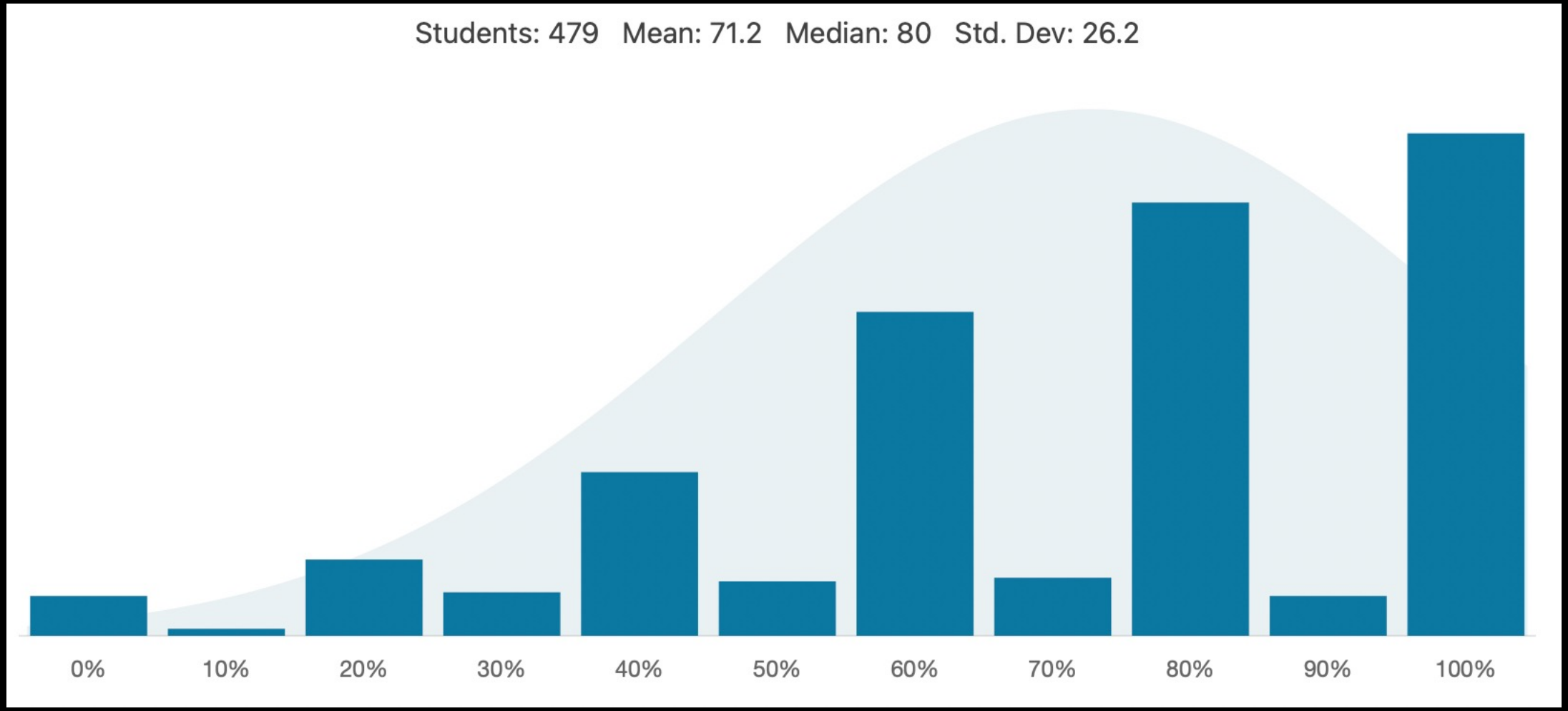
Question 1


Students: 479 Mean: 80.8 Median: 80 Std. Dev: 16.7
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

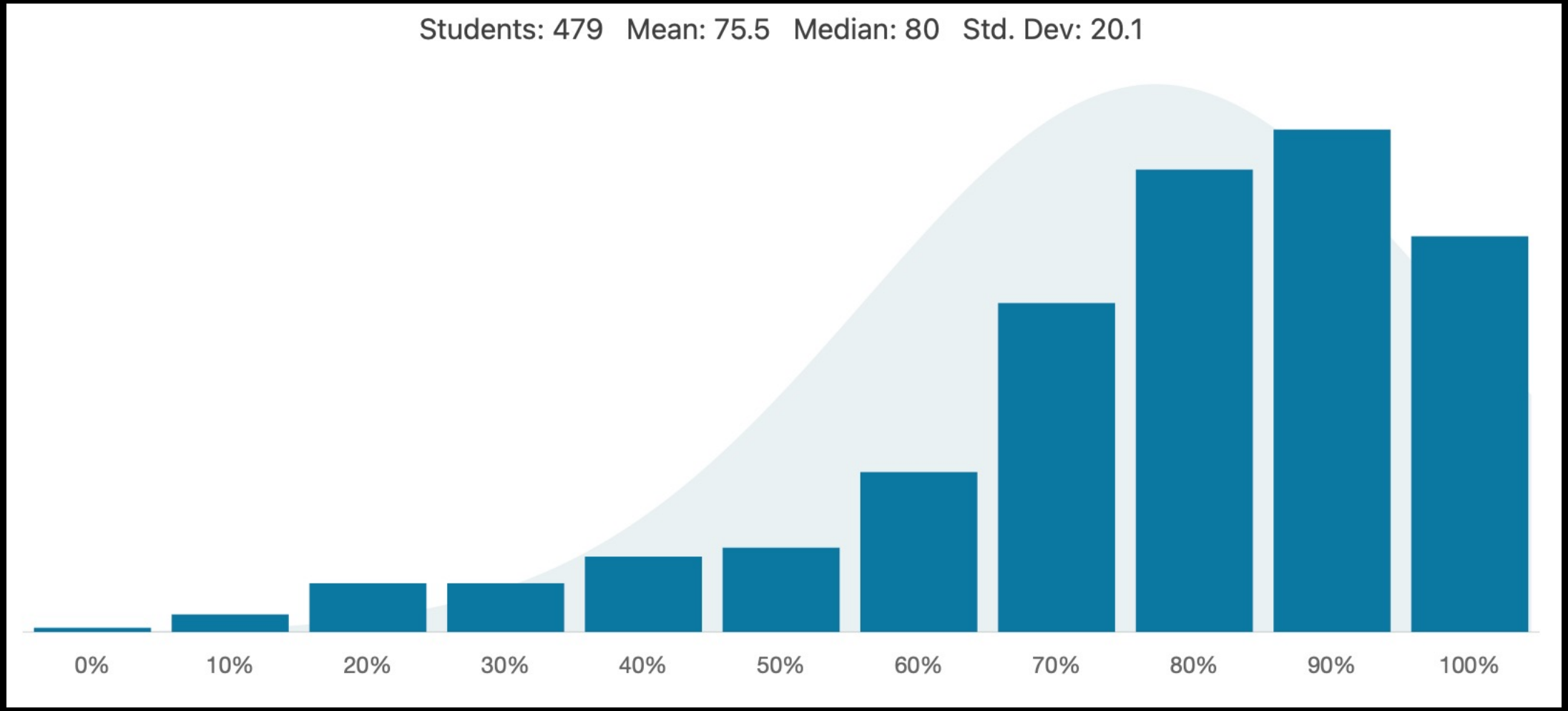
Question 2


Students: 479 Mean: 54.4 Median: 60 Std. Dev: 34.4
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

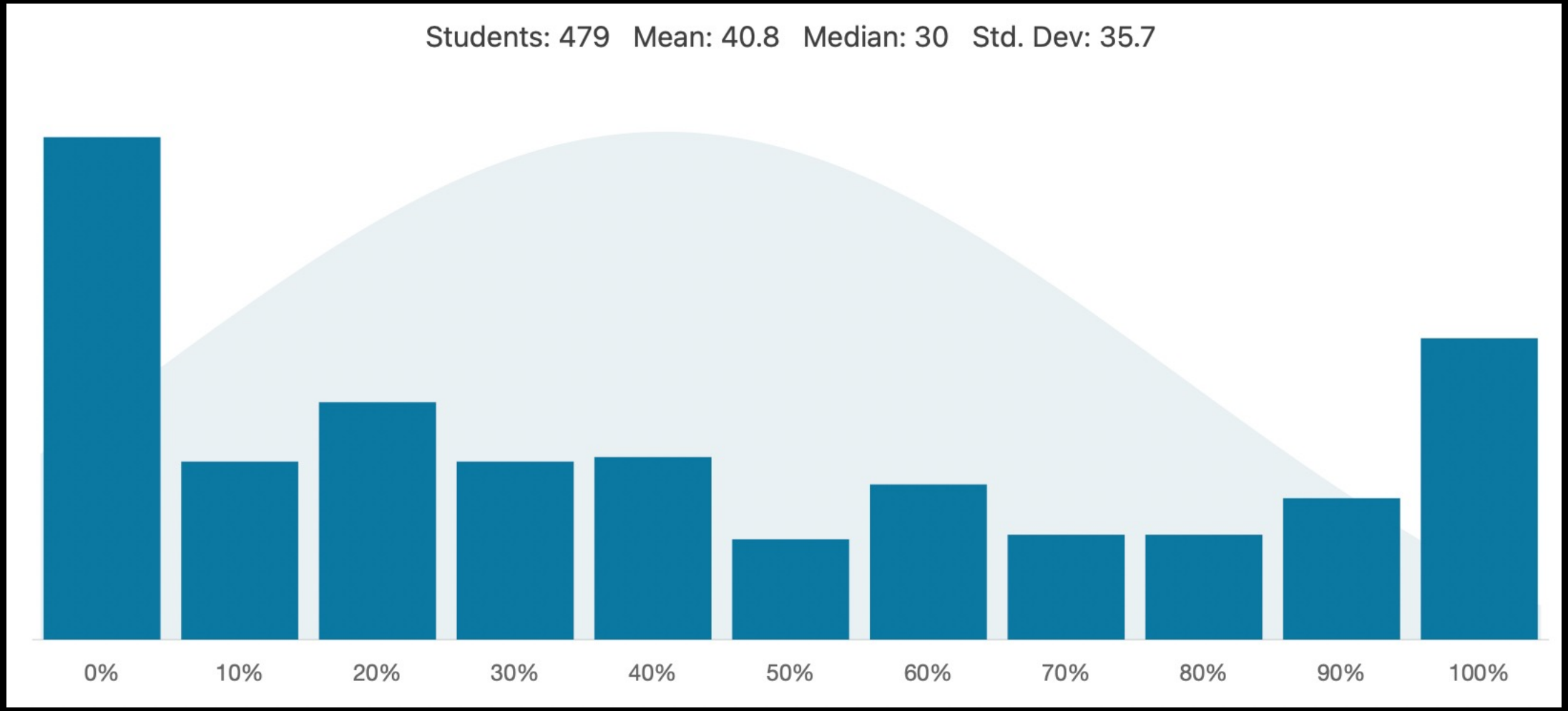
Question 3


Students: 479 Mean: 71.2 Median: 80 Std. Dev: 26.2
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

Question 4


Students: 479 Mean: 75.5 Median: 80 Std. Dev: 20.1
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

Question 5


Students: 479 Mean: 40.8 Median: 30 Std. Dev: 35.7
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%

Next Midterm: March 15

Motivation

We want to keep track of characters in a complex show/book

- ✓Name
- ✓Actor
- ✓Personality
- ✓Age
- ✓Title/Powers

- We could store values in a string?
- We could have unique variable names for each person?

```
gandalf_age = 24000
frodo = "Frodo-Elijah Wood-brave, observant, and unfailingly polite-51-Ring bearer"
```

We need an efficient way to do this.

One way: Tables or Lists!

Name	Gender	Actor	Personality	Age	Powers
Sam	M				
Frodo	M				
Gandalf	M				
Galadriel	F				
Pippin	M				
Aragorn	M				
Legolas	M				
Eowyn	F				
Gollum	M				
Arwen	F				
Merry	M				

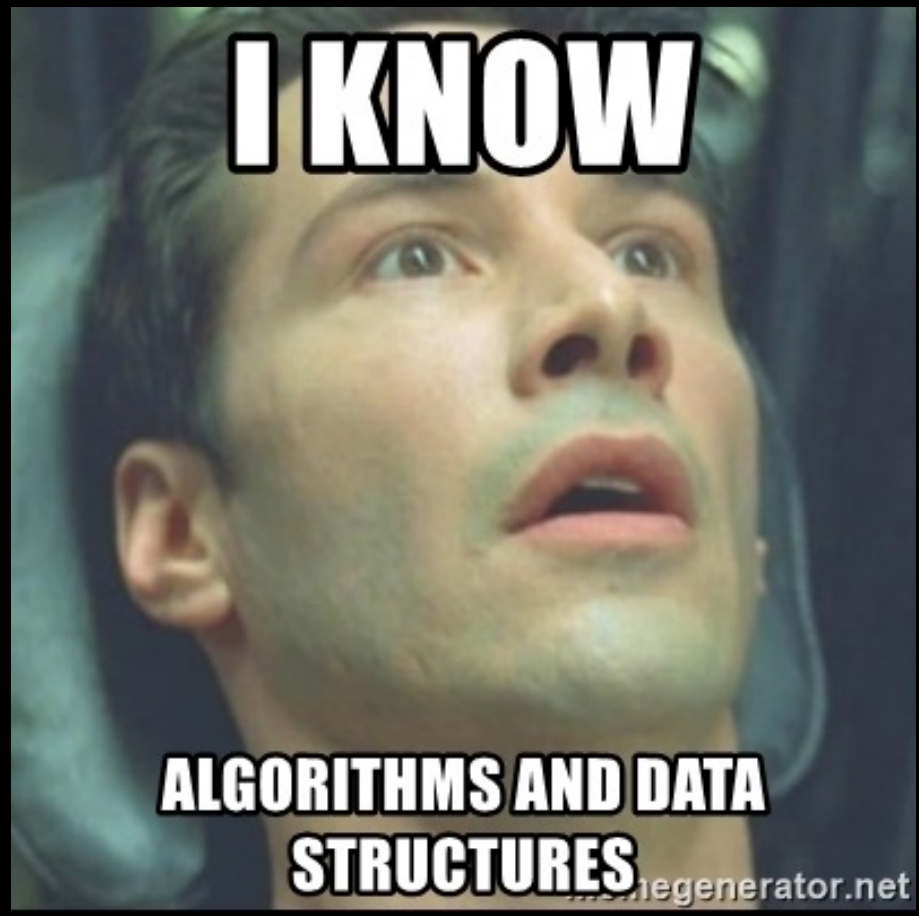
Need to:

- ✓Create rows of data
- ✓Create columns of data
- ✓Be able to access a specific cell/index

Data Structures!

Data structures are "containers" that organize and group data

- Lists
- Sets
- Tuples
- Dictionaries
- Linked lists
- Binary trees



Type: List

- Can store an **ordered** collection of data using Python's type list
- The general form of a list is:

```
[val1, val2, val3, …, valN]
```

- Values are enclosed in ([ ]) and separated by commas (,)
- Can assign lists to a variable name:

```
my_list = [val1, val2, val3, …, valN]
```

List Elements

- list elements can be of any type:

```
subjects = ['bio', 'programming', 'math', 'history']
grades = [75, 98, 82, 62]
```

- A list can contain elements of more than one type:

```
street_address = [10, 'Main Street']
light = ['status',  True, 'intensity', 3.1]
```

List Operations (Indexing and Slicing)

- A list can be indexed just like a string:

```
>>> grades = [80, 90, 70, 45, 98, 57]
>>> grades[1]
90
>>> grades[-3]
45
```

- A list can be sliced just like a string:

```
>>> grades[0:2]
[80, 90]
```

```
>>> grades[::-2]
[57, 45, 90]
```

Nested Lists

- Lists can contain any type, including other lists!
  - Called "nested lists"

```
[list1, list2, …, listN]
```

```
[val1, val2, …, valN]
```

- To access a nested item, first select the sublist, then treat as a regular list

```
>>> list_of_lists[0]
[val1, val2, …, valN]
>>> list_of_lists[0][1]
val2
```

Nested Lists Example

- Let's provide some information in our list of grades:

```
>>> aps106_grades = [['Midterm 1', 60],['Midterm 2', 90],['Exam', 100]]

>>> aps106_grades = [['Midterm 1', 60],
                     ['Midterm 2', 90],
                     ['Exam', 100]]
```

BOTH OF THESE ARE THE SAME THING!

- Now we can access different parts depending on what we want:

```
>>> aps106_grades[0]
['Midterm 1', 60]

>>> aps106_grades[2][1]
100
```

Let's Code!

- Let's take a look at how this works in Python!
  - Creating lists
  - List indexing and slicing
  - List operations
  - Nested lists!

Open your notebook

Click Link:
1. The 'list' Type

List Mutability

- Lists are mutable!
  - This means they can be mutated (modified)

- All the other types we've learned so far (string, int, float, and bool) are immutable (i.e. they can NOT be modified)

List Mutability Example

strings are immutable

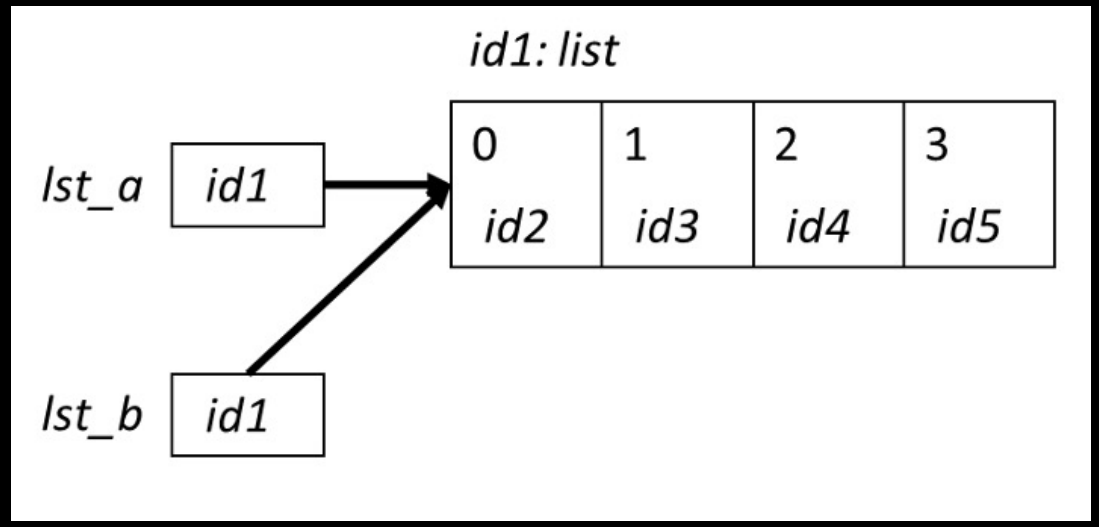
```
>>> s = "I love cats"
s[0] = "U"
Traceback (most recent call last):
builtins.TypeError: 'str' object does not
support item assignment
```

lists are mutable

```
>>> grades = [80, 90, 70, 45, 98, 57]
>>> grades[3] = 100
>>> grades[-1] = 100
>>> grades[2] = 'Perfect'
>>> grades
[80, 90, 'Perfect', 100, 98, 100]
```

Aliasing

- When two variable names refer to the same object, they are aliases.
- When we modify one variable, we are modifying the object it refers to, hence also modifying the second variable.



- This is common source of error when working with list objects.

Aliasing Example (with Visualizer)

Permalink:

https://pythontutor.com/visualize.html#code=str1%20%3D%20%22abcd%22%0Astr2%20%3D%20str1%0Astr1%20%3D%20str1%2B%22e%22%0A%0Alst1%20%3D%20%5B11,%2012,%2013,%2014,%2015,%2016,%2027%5D%0Alst2%20%3D%20lst1%0Alst1%5B-1%5D%20%3D%2018&cumulative=false&curInstr=0&

Avoiding Aliasing

```
>>> lst1 =  [11, 12, 13, 14, 15, 16, 27]
>>> lst2 = lst1
>>> lst1[-1] = 17
>>> lst2
[11, 12, 13, 14, 15, 16, 17]

>>> id(lst1)
49012568
>>> id(lst2)
49012568
```

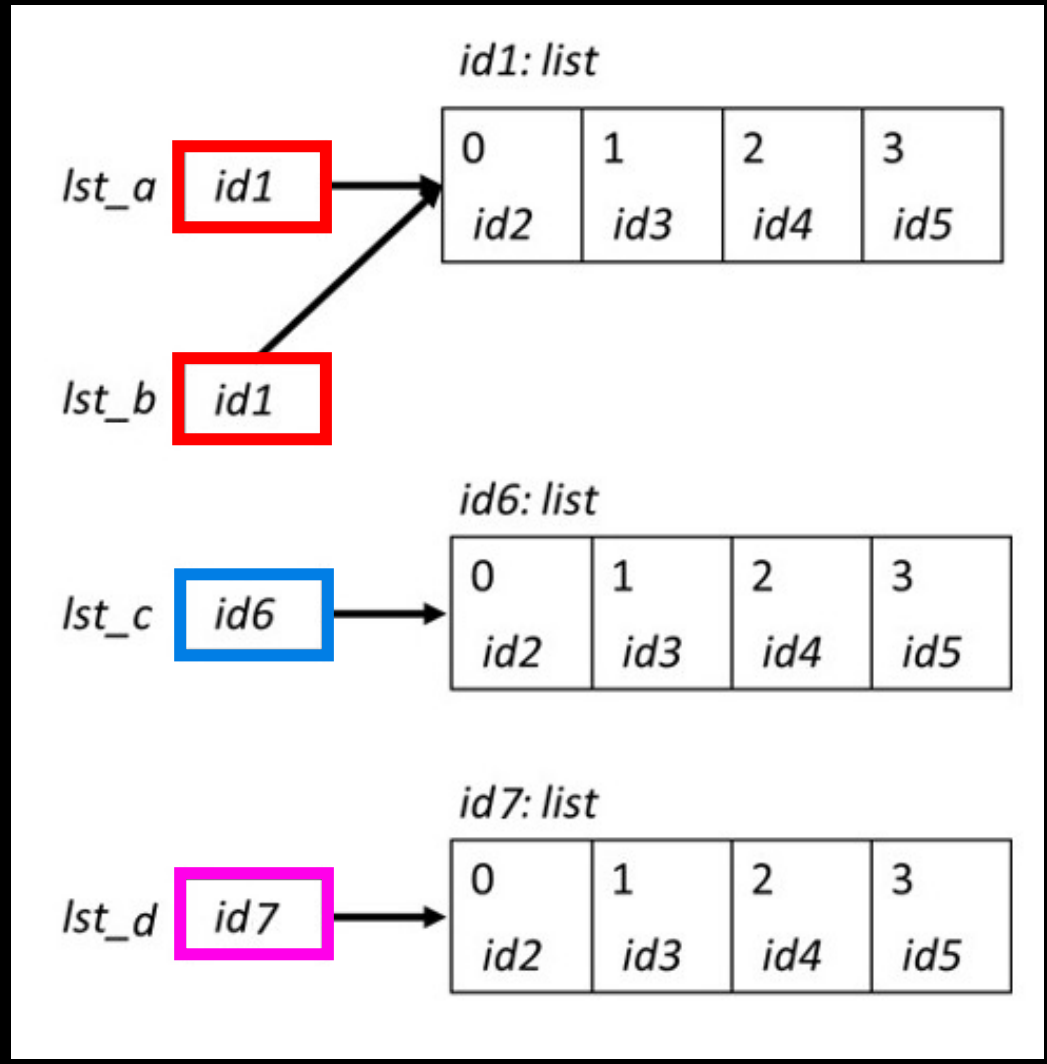
- How can we copy lst1 into another list without aliasing?

Copying Lists and Avoiding Aliasing

- There are two simple ways to copy lists:
  - Using the `list( )` function
  - Completely slice the list `[:]`

```
>>> lst_a =  [0, 1, 2, 3]
>>> lst_b = lst_a
>>> lst_c = list(lst_a)
>>> lst_d = lst_a[:]

>>> id(lst_a)
39012510
>>> id(lst_b)
39012510
>>> id(lst_c)
54514112
>>> id(lst_d)
24514139
```



Avoiding Aliasing Example (with Visualizer)

Permalink:

https://pythontutor.com/visualize.html#code=lst1%20%3D%20%5B11,%2012,%2013,%2014,%2015,%2016,%2027%5D%0Alst2%20%3D%20lst1%0Alst1%5B-1%5D%20%3D%2017%0A%0Alst3%20%3D%20list%28lst1%29%0Alst4%20%3D%20lst1%5B%3A%5D&cumulative=false&curInstr=0&heapPrimitives=false&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false

Let's Code!

- Let's take a look at how this works in Python!
  - List mutability
  - Aliasing
  - Copying lists

Open your notebook

Click Link:
2. Mutability and Aliasing

Built-in Functions

- Several of Python's built-in functions can be applied to lists, including:
  - `len(list)` : return the number of elements in list (i.e. the length)
  - `min(list)` : return the value of the smallest element in list.
  - `max(list)` : return the value of the largest element in list.
  - `sum(list)` : return the sum of elements of list (list items must be numeric).

List Methods

- Lists are objects and just like other objects, the list type has associated methods that are only valid for lists
- Recall you can find out which methods are associated with objects using the built-in function dir

```
>>> dir(list)
['__add__', '__class__', '__class_getitem__', '__contains__',
'__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__',
'__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Adding Items to a List

- To add an object to the end of a list, use the list method append:

```
>>> colours = ['blue', 'yellow’]
>>> colours.append('brown’)
>>> colours
['blue', 'yellow', 'brown']
```

- To add a list to the end of a list, use the list method extend:

```
>>> colours = ['blue', 'yellow’]
>>> colours.extend([‘pink’,‘green’])
>>> colours
['blue', 'yellow', 'brown’, ‘pink’, ‘green’]
```

Removing Items from a List

- To remove an object from a list, use the list method remove:

```
>>> colours = ['blue', 'yellow', 'pink']
>>> colours.remove('yellow')
>>> colours
['blue', 'pink']


>>> colours.remove('red')
Traceback (most recent call last):
builtins.ValueError: list.remove(x): x not in list
```

How can we write it so there's no error?

Is something in my list?

- The in operator can be used on lists too!

```
colours = ['blue', 'yellow', 'pink']

if 'red' in colours:
    colours.remove('red')
```

Let's Code!

- Let's take a look at how this works in Python!

Open your notebook

Click Link:
3. List Methods

APS106

UNIVERSITY OF TORONTO

Lists: indexing and slicing.

Week 7 | Lecture 1 (7.1)

if nothing else, write #cleancode