Announcements

- HW5 graded and uploaded back to Github
- Midterm postponed!
  - In light of everything, we are moving it back until the Friday you get back
  - Will have a homework that week, so we'll figure out what that will look like.
- Normal class day on Friday
- No lab today!
- I'll be getting updated grade-reports out during Spring Break
- Polling: `rembold-class.ddns.net`

Review Question

Which of the provided options would appear as below when printed? The sideways brackets are JUST to show you spaces. They would not appear!

101,234.98␣␣␣&␣4000

A) `'{:<12,f} & {:0>4d}'.format(1.01234984e5, 320//8)`

B) `'{>12,.2f} & {0>4d}'.format(1.01234984e5, 32000//8)`

C) `'{:<12,.2f} & {:<4d}'.format(1.01234984e5, 3200//8)`

D) `'{:<12,.2f} & {:0<4d}'.format(1.01234984e5, 32//8)`

Solution: `'{:<12,.2f} & {:0<4d}'.format(1.01234984e5, 32//8)`

Ain't no g-string

- Short for format string, an f-string achieves the same thing as `.format` but with less syntax
- Introduced in Python 3.6
- Need an 'f' right before the string to let Python know it needs to do more with the string
- Place the desired variables (or values) directly into the `{ }`, where you'd normally have placed the label!

```
A = 10
B = 15.123234
print(f'A is {A} and B is {B:.2f}')
```

- All other syntax and format specs like `.format`!

Objectively...

- We've discussed many different kinds of data so far:

```
123     3.14     'Hello'     [2,4,6]
True    (3,5,2)    {'Apples':3, 'Bananas':6}
```

Objectively...

- We've discussed many different kinds of data so far:

```
123     3.14     'Hello'     [2,4,6]
True    (3,5,2)    {'Apples':3, 'Bananas':6}
```

- Each is an object, and every object has:
  - a *type*
  - some form of internal *data representation*
  - a set of ways to *interact* with the object

Objectively...

- We've discussed many different kinds of data so far:

```
123     3.14     'Hello'     [2,4,6]
True    (3,5,2)    {'Apples':3, 'Bananas':6}
```

- Each is an object, and every object has:
  - a *type*
  - some form of internal *data representation*
  - a set of ways to *interact* with the object
- More particularly, an object is an instance of some *type*
  - `True` is an instance of type `bool`
  - `3.14159` is an instance of type `float`

All the Objects

- In Python, everything is an object (and thus an instance of some type)
  - Can create new objects of some type
    - `A = 'hello'`
  - Can manipulate objects
    - `A.lower()`
  - Can destroy objects
    - Can use `del` or reassign their label
    - Python will clean up deleted or inaccessible objects
    - Called garbage collection

What's my type?

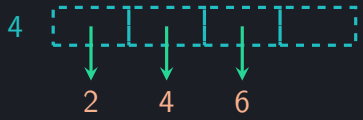
- When you think about it, a type in Python encompasses two main ideas:
  1. It defines what the objects of that type *are*. What properties it has. How it is represented internally in memory.

What's my type?

- When you think about it, a type in Python encompasses two main ideas:
  1. It defines what the objects of that type *are*. What properties it has. How it is represented internally in memory.
  2. It defines an *interface* for how one interacts with objects of that type and how objects of that type interact with other objects.

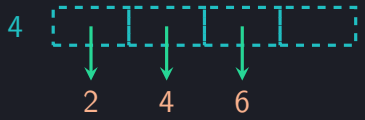
Example: Lists

- Consider the object `[2,4,6]` which has type `list`
- Has an internal representation of two parts:
  - A list of pointers to each of the element objects
  - A value that keeps track of the allocated memory for the above list



Example: Lists

- Consider the object `[2,4,6]` which has type `list`
- Has an internal representation of two parts:
  - A list of pointers to each of the element objects
  - A value that keeps track of the allocated memory for the above list



- Has ways to manipulate lists:
  - `L[i]`, `L[i:j]`
  - `len()`, `max()`, `del(L[i])`
  - `L.append()`, `L.count()`, `L.remove()`, etc
  - We use these methods instead of messing with the internal representation

Object Oriented Programming

- Object Oriented Programming is just a paradigm of making objects and their accompanying types the star of the show
- Python is setup nicely for this paradigm considering everything is already an object
- Advantages:
  - Bundles data and procedures to operate on that data together in nice packages and with well defined interfaces
  - Allows to implement and test different object types independently
  - Modularity reduces complexity and makes it easy to reuse code

Relation to Functions

- Python already has a large list of built in functions
- Often need or could benefit from custom functions though, so we learned how to define our own
  - Had the definition portion, where we established what the function does, accepts as inputs, and returns
  - Had the function call, where we actually utilize the function

Relation to Functions

- Python already has a large list of built in functions
- Often need or could benefit from custom functions though, so we learned how to define our own
  - Had the definition portion, where we established what the function does, accepts as inputs, and returns
  - Had the function call, where we actually utilize the function

Now we want to do the same for **types**!

Getting classy

- We define a new type in Python by defining a new class
- Classes and types are synonymous in Python 3
- Like functions, will have two "parts":
  - A definition portion, where we define what the type is and how we can interact with it
  - An instance portion, where we create new objects of our type to utilize in our code
    - "An instance of C" and "An object of type C" are telling you the same thing.