What's the largest possible value that Python's `int` type can store?

Answer: There is no largest value - the int type dynamically adapts the memory it needs.

This also means that if you use larger values, the int type will become slow.

What's the best built-in type in Python to use for representing monetary amounts?

Answer: `int`.

Of the types mentioned in the question, `int` is the most suitable. The reason you don't want to use `float` is that they're not precise enough to store monetary amounts. Also, monetary amounts always have some kind of base unit (like cents or satoshi) that you can't divide any further ("half a dollar cent" is not a meaningful amount). As a result, types that support fractions (like `float`) need to be rounded in order to represent a valid monetary amount. The nice thing about `int` is that rounding happens automatically, assuming you represent monetary amounts in the base units. So $100 stored in an `int` variable wouldn't be `100`, but `10000` (10,000 cents). Integers are commonly used in this way in many payment systems, including Stripe, one of the largest payment providers in the world.

Alternatively, you can use the `Decimal` type for monetary values because it's more precise than `float`. However, you'd still need to deal with rounding at the end. Therefore, the best approach is probably to use `Decimal` in computations for precision, but store final monetary amounts as integers.

Why is int better suited for representing monetary values than float?

Answer: The float type is not precise enough for representing monetary values.

(see the discussion above)

What happens when you try to add a `float` and an `int` in Python?

The result is computed and is of type `float`. Even though Python is strongly typed, numerical types are related and can be used in the same expression without type conversion.

What do we mean if we say that a type is *mutable*?

Answer: Objects of a mutable type are allowed to change their value.

This is as opposed to *immutable* types, which are not allowed to change their value (like `str`). Examples of mutable types in Python are `list` and `dict`.

Consider the following lines of code:

```
my_list = [0, 2, 4, 6, 7, 5]
new_list = my_list[4:] + my_list[:2]
```

What's the value of `new_list`?

Answer: `[7, 5, 0, 2]`

The expression `my_list[4:]` returns the sublist of `my_list` starting at index 4. The expression `my_list[:2]` returns the sublist of `my_list` starting at index 0 and ending at index 2. The expression `my_list[4:] + my_list[:2]` returns the concatenation of the two sublists.

For a web scraping project, you need to cache search results based on keywords. What's the most suitable data structure to store these cached results in?

Answer: `dict`.

The reason is that you want to be able to retrieve cached results based on keywords. If you use a `list`, you'd have to search linearly through the list to find the cached result (in the worst case). With a `dict`, you can look up the cached result by keyword resulting in a much faster search.

An image processing pipeline needs a data structure to represent the sequence of processing operations to be performed on a particular image. What's the best data structure for this?

Answer: `list`.

A pipeline is a squence of operations, so we need a data structure that allows for this kind of ordered representation natively. A list is a natural choice here.

The function `connect_to_database` can setup a connection to a database of choice, and it returns a `Connection` object with which you can access the database as well as a `ConnectionResult` object that contains information about the kind of connection that was established. What's the most suitable data structure for containing these two objects?

Answer: `tuple`.

A tuple is useful for representing a (small) collection of object of different types. This is different from a list, dict, or set, which generally are containers of objects of the same type.

A profanity filter has a collection of profane words that it uses to filter user messages. What's the most suitable data structure for this word collection?

Answer: `set`.

A set is the most natural choice here, because we don't need to store duplicates and we probably mainly want to do lookups to determine if a word is in the set.

I want to build a system where, given a customer's (unique) email address, I can quickly retrieve the customer information. I already have a class `Customer` that contains all that information (including the email address). What's the most performant approach?

Answer: Create a `dict[str, Customer]` and use the email address as a key.

The dictionary is a very good choice since it allows you to link unique keys (like an email address) to objects. Because the dictionary uses hash tables under the hood, it's very fast to look up keys.