Updates from the original design

Concrete syntax for match arms

We replaced the `end` at the end of each `match` arm with a comma `,`. The comma is not needed at the end of the last arm.

Shadowing primitive types

We added checks in `check_prog` to ensure that shadowing primitive types is not permitted.

```
type Num in ...
```

```
let Bool = 1 in ...
```

```
def Array(): 1 in ...
```

These will all trigger an error in compile time.

Duplicate field names in destructor

When destructuring a custom type value, we do not allow the same field name to be used twice:

```
type Ginger(a, b) in
match 0 default 0:
  case Ginger(a, a) => ...
end
```

However, this is still valid:

```
type Ginger(a, a) in ...
```

Heap layout

Rather than putting the actual fields array inside a custom type value, we decided to just put a pointer. The pointer is normally tagged and is no different from a standard Snake value representing an array.

```
|          type tag          |
|      pointer to array      | <-- points to
                                  |      number of fields     |
                                  |          field 1          |
                                  |          field 2          |
                                  |            ...            |
```

Pretty printing

Rather than printing the type tag for custom type values, we implemented pretty printing. Note some additional requirements: `runtime/custom_types.txt` must be writable, and the compiler must be invoked in the project root directory.

```
type Ginger(size), Fish in
print(Ginger(2));
Fish
```

will print

```
Ginger(2)
Fish
```

instead of

```
<0 : [2]>
<1 : []>
```