CMPU 101 § 1 · Problem-Solving and Abstraction

Tables and Helper Functions

13 September 2021

Helper functions

We write functions to avoid repeating the same computations over and over, as we saw with our cake shop functions.

But what if you find yourself repeating code between functions?

Consider:

```
fun greet(firstname, surname, position):
  if position == "professor":
    string-append("Hello Professor ", surname)
  else if position == "student":
    string-append("Hello ", firstname)
  else:
    string-append("Hello ",
      string-append(firstname,
        string-append(" ", surname)))
  end
end
```

and

```
fun leave(firstname, surname, position):
  if position == "professor":
    string-append("Bye Professor ", surname)
  else if position == "student":
    string-append("Bye ", firstname)
  else:
    string-append("Bye ",
      string-append(firstname,
        string-append(" ", surname)))
  end
end
```

We're solving the same problem – figuring out how to address a person based on their name and position – twice.

And we might need to write many more functions that need to do the same thing!

```
fun greet(firstname, surname, position):
  string-append("Hello ",
    name(firstname, surname, position))
end

fun leave(firstname, surname, position):
  string-append("Bye ",
    name(firstname, surname, position))
end

fun name(firstname, surname, position):
  if position == "professor":
    string-append("Professor ", surname)
  else if position == "student":
    firstname
  else:
    string-append(firstname,
      string-append(" ", surname))
  end
end
```

Big functions cause lots of problems!

Helper functions let us keep our functions small, readable, and testable.

Write as many helper functions as a problem seems to require, even if an assignment or lab doesn't explicitly tell you to!


Functions,
Functions who need functions,
Are the luckiest functions in the world…

Example: Gradebook

We've now had two labs, our first assignment is out, the grading is piling up.

Let's make my life easier by writing a program to manage a gradebook.

A central problem is computing various averages, e.g.,

the average of how everyone does on Assignment 1, or

the average of a student across all their assignments.

Let's say:

- Allie gets
  - 85% on Assignment 1
  - 90% on Assignment 2
- Carl gets
  - 75% on Assignment 1
  - 60% on Assignment 2

How can a function look up the grade a student gets on a specific assignment?

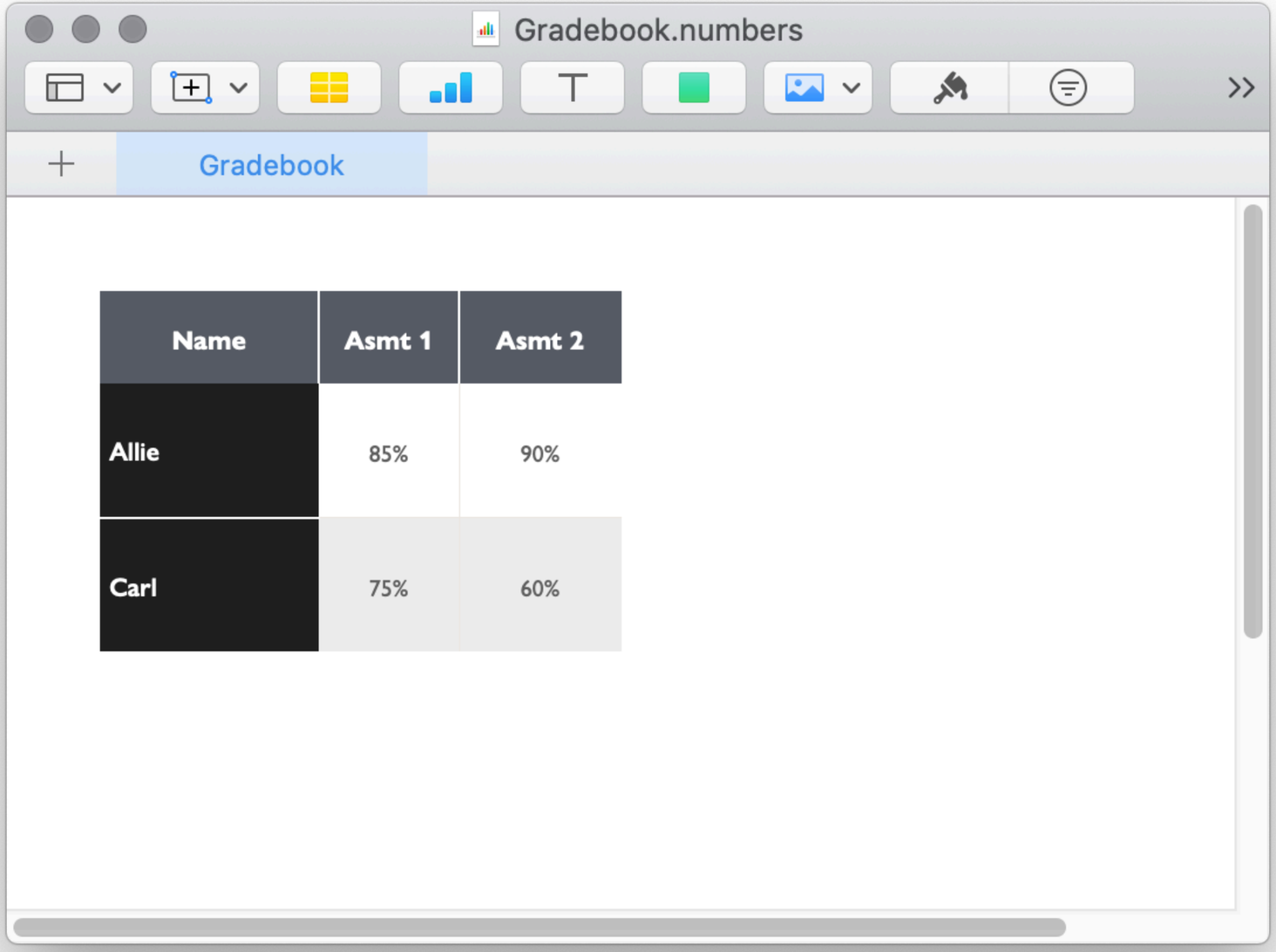
```
fun look-up-grade(student :: String, asmt :: String) -> Number:
  doc: "Return grade of a given student on a given assignment"
  if student == "Allie":
    if asmt == "asmt1":
      85
    else if asmt == "asmt2":
      90
    else:
      raise("No such assignment")
    end
  else if student == "Carl":
    if asmt == "asmt1":
      75
    else if asmt == "asmt2":
      60
    else:
      raise("No such assignment")
    end
  else:
    raise("No such student")
  end
end
```

This is not a great way to do this.

Why not?

KEY IDEA Separate data from computations.

In practice, how do instructors keep gradebooks?



Name	Asmt 1	Asmt 2
Allie	85%	90%
Carl	75%	60%

Tables

```
gradebook = table: name, NRO, asmt1, asmt2
  row: "Allie", false, 85, 90
  row: "Carl",  false, 75, 60
  row: "Elan", true, 95, 63
  row: "Lavon", false, 87, 88
  row: "Nunu", true, 70, 0
end
```

```
gradebook = table: name, NRO, asmt1, asmt2
  row: "Allie", false, 85, 90
  row: "Carl",  false, 75, 60
  row: "Elan", true, 95, 63
  row: "Lavon", false, 87, 88
  row: "Nunu", true, 70, 0
end
```

What computations might you want to do with this table?

We could

- Compute course grades
- Get a histogram of performance on each assignment
- Look at a student's change (delta) from the first assignment to the second assignment
- Check whether students who NRO did worse on Assignment 2 than those who didn't
- Get names of students who did poorly on the first assignment
- And more

What operations do you need to do these things?

To look only at low grades, you need to

Filter out some rows,

To see high or low scores first, you need to

Re-order the rows,

To compute the average for students who NRO or don't, you need to

Perform computation based on a particular column, and

To compute the average for each student, you might want to

Add a new column with particular values.

```
gradebook = table: name, NRO, asmt1, asmt2
   row: "Allie", false, 85, 90
   row: "Carl",  false, 75, 60
   row: "Elan", true, 95, 63
   row: "Lavon", false, 87, 88
   row: "Nunu", true, 70, 0
end
```

KEY IDEA Once data are made up of smaller pieces of data, we want to organize the data to make it easier to maintain and process.

Tables are good for data about multiple entities, each of which has the same attributes.

```
gradebook = table:
  name :: String,
  NRO :: Boolean,
  asmt1 :: Number,
  asmt2 :: Number
  row: "Allie", false, 85, 90
  row: "Carl",  false, 75, 60
  row: "Elan", true, 95, 63
  row: "Lavon", false, 87, 88
  row: "Nunu", true, 70, 0
end
```

As with functions, we can specify the types for parts of a table.

Functions over tables

To have all the functions we want for working with tables, let's use a library:

```
include shared-gdrive("dcic-2021",
"1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
```

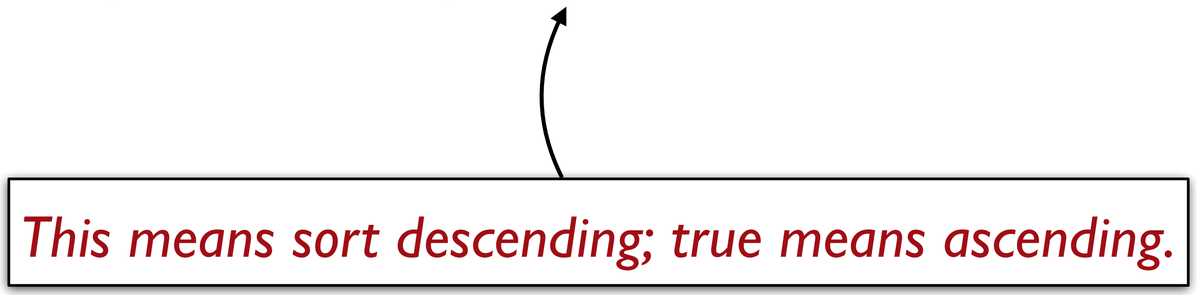
Order the rows by descending values on Assignment 1:

Order the rows by descending values on Assignment 1:

```
order-by(gradebook, "asmt1", false)
```

Order the rows by descending values on Assignment 1:

```
order-by(gradebook, "asmt1", false)
```



```
order-by(t :: Table,
  colname :: String,
  sort-up :: Boolean)
  -> Table
```

Given a table and the name of a column in that table, return a table with the same rows but ordered based on the named column.

If **`sort-up`** is `true`, the table will be sorted in ascending order, otherwise it will be in descending order.

Keep only the rows in which the NRO column contains true.

The **`filter-with`** function produces a table with rows for which a given function returns **`true`**:

```
filter-with(gradebook, taking-nro)
```

So, we need a function that takes a row and produces a Boolean indicating whether to keep the row:

```
fun taking-nro(r :: Row) -> Boolean:
  doc: "Get the value in the given row's NRO column"
  r["NRO"]
end
```

And another for those who aren't NRO-ing:

```
fun not-taking-nro(r :: Row) -> Boolean:
  not(r["NRO"])
end
```

```
filter-with(t :: Table,
  keep :: (Row -> Boolean))
  -> Table
```

Given a table and a predicate on rows, returns a table with only the rows for which the predicate returns `true`.

Keep those students whose grades dropped from Assignment 1 to Assignment 2:

```
fun asmt2-lower(r :: Row) -> Boolean:
  r["asmt1"] > r["asmt2"]
end

filter-with(gradebook, asmt2-lower)
```

To get just the first row from the table, we use its numeric index:

```
gradebook.row-n(0)
```

To get a particular column's value from a row, we specify the column name using square brackets:

```
gradebook.row-n(0)["asmt1"]
```

Acknowledgments

This lecture incorporates material from:

Kathi Fisler, Brown University