R programming basics: coding etiquette

Ecological Systems Modeling

Jan 15-19, 2024

Active participation (optional)

- Open RStudio in Jupyter Hub
- In the Files/Plots/etc. pane, navigate to: `$HOME/Labs/Intro_R_part2/`
- Click on `Intro_to_R_part2c_codingEtiquette.rmd`
- File should open in the Source pane
- Run the code chunks, add to chunks, or type code in Console

Learning objectives

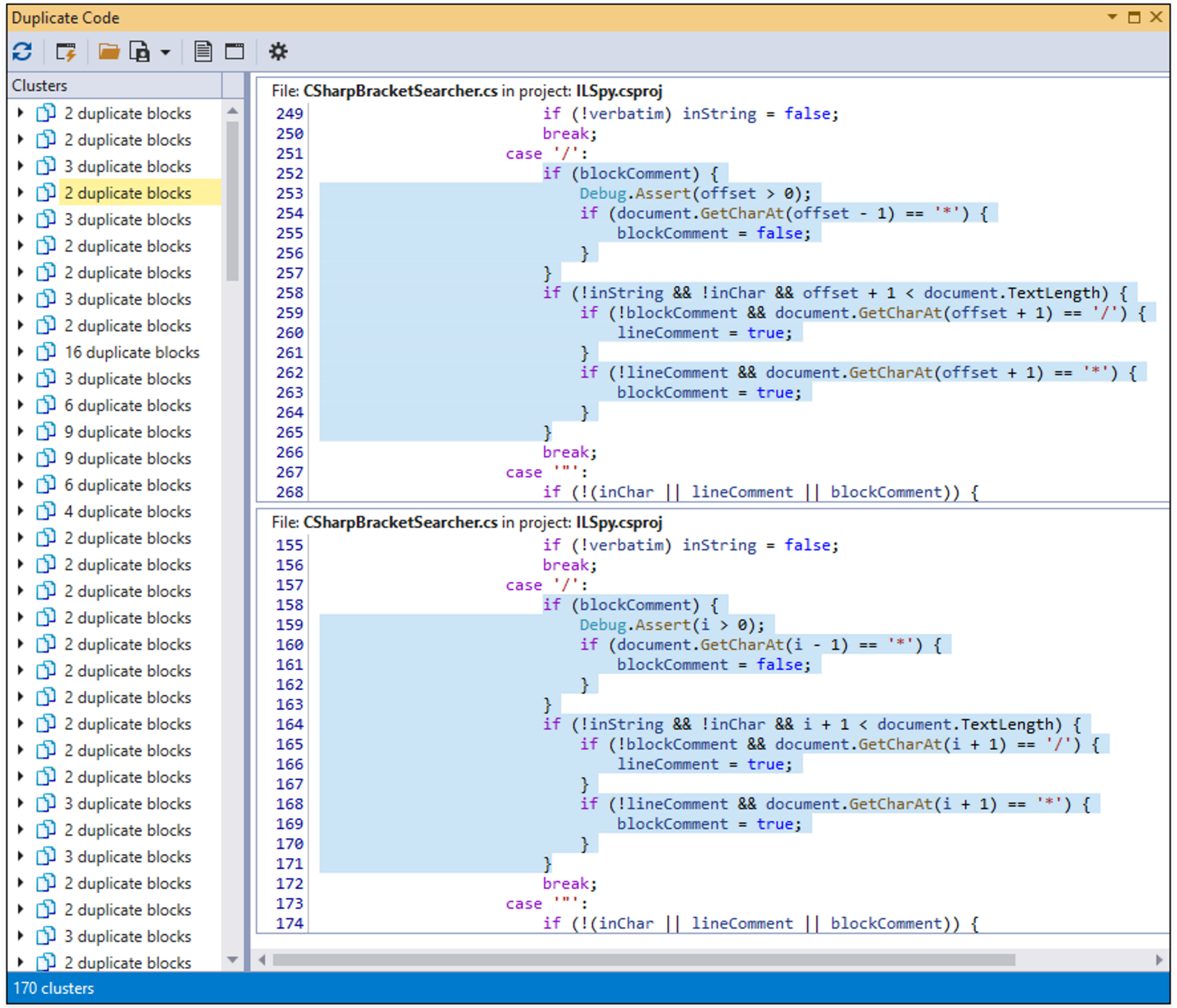
- Describe several types of best coding practices
- Apply this knowledge in your code for this course
  - Lab assignments
  - Class project

Overview: best coding practices

1. Abide by the DRY (Don’t Repeat Yourself) principle
2. Follow some easy-to-remember naming convention
3. Keep the code as straightforward as possible
4. Limit the length of a line of code
5. Use comments frequently
6. Use consistent indentation
7. Whenever and wherever possible, avoid deep nesting

1. DRY: Don't Repeat Yourself

- Don't write the same code repeatedly
- Example shows duplicate C# code
- Instead, write and re-use functions (more on this later)


Duplicate Code

Clusters

```
2 duplicate blocks
2 duplicate blocks
3 duplicate blocks
2 duplicate blocks
3 duplicate blocks
2 duplicate blocks
2 duplicate blocks
3 duplicate blocks
2 duplicate blocks
16 duplicate blocks
3 duplicate blocks
6 duplicate blocks
9 duplicate blocks
9 duplicate blocks
6 duplicate blocks
4 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
2 duplicate blocks
3 duplicate blocks
2 duplicate blocks
3 duplicate blocks
2 duplicate blocks
2 duplicate blocks
3 duplicate blocks
2 duplicate blocks
```

File: **CSharpBracketSearcher.cs** in project: **ILSpy.csproj**

```
        if (!verbatim) inString = false;
        break;
    case '/':
        if (blockComment) {
            Debug.Assert(offset > 0);
            if (document.GetCharAt(offset - 1) == '*') {
                blockComment = false;
            }
        }
        if (!inString && !inChar && offset + 1 < document.TextLength) {
            if (!blockComment && document.GetCharAt(offset + 1) == '/') {
                lineComment = true;
            }
            if (!lineComment && document.GetCharAt(offset + 1) == '*') {
                blockComment = true;
            }
        }
        break;
    case '"':
        if (!(inChar || lineComment || blockComment)) {
```

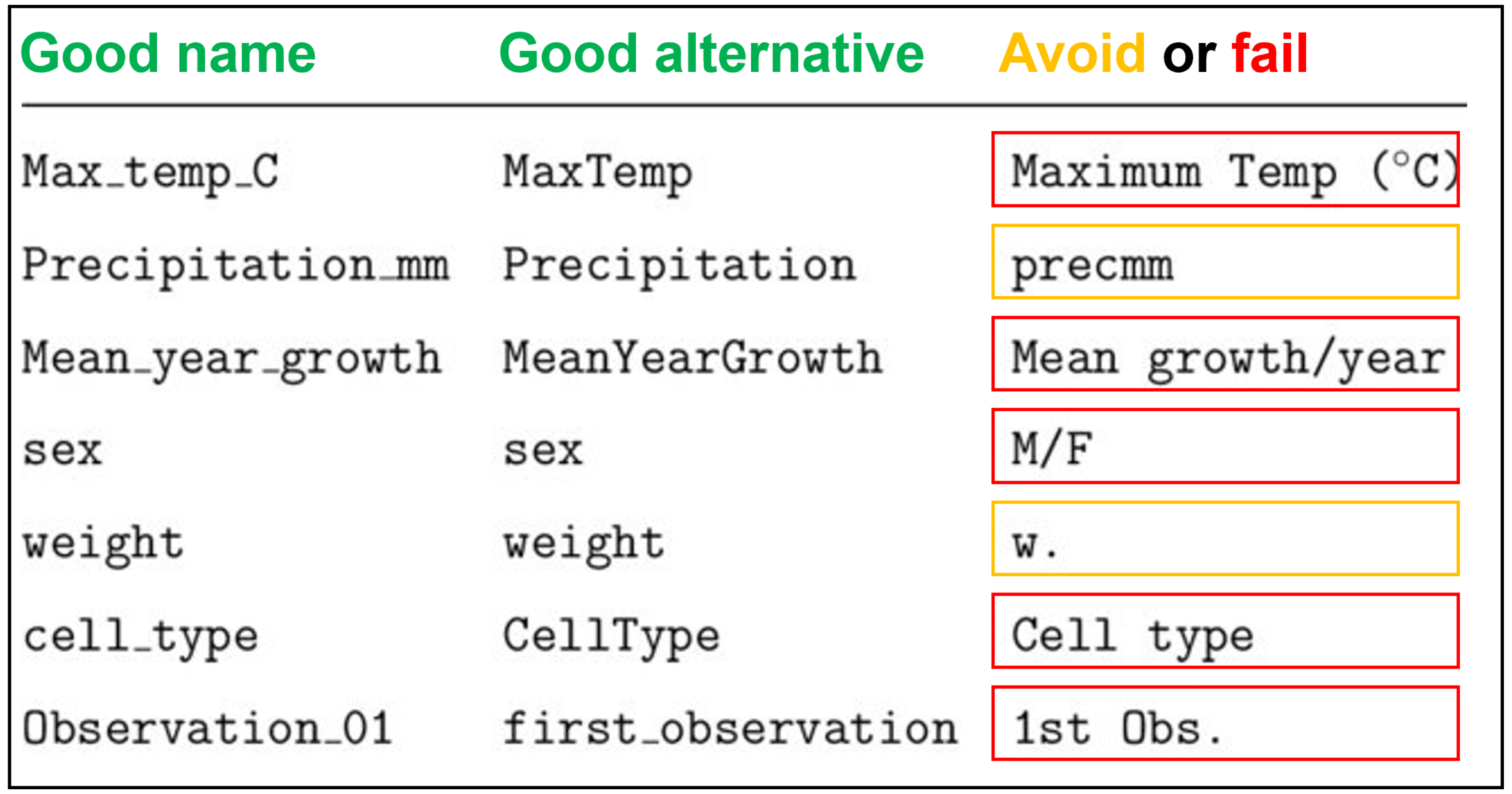
File: **CSharpBracketSearcher.cs** in project: **ILSpy.csproj**

```
        if (!verbatim) inString = false;
        break;
    case '/':
        if (blockComment) {
            Debug.Assert(i > 0);
            if (document.GetCharAt(i - 1) == '*') {
                blockComment = false;
            }
        }
        if (!inString && !inChar && i + 1 < document.TextLength) {
            if (!blockComment && document.GetCharAt(i + 1) == '/') {
                lineComment = true;
            }
            if (!lineComment && document.GetCharAt(i + 1) == '*') {
                blockComment = true;
            }
        }
        break;
    case '"':
        if (!(inChar || lineComment || blockComment)) {
```

170 clusters

2. Easy-to-remember naming convention

- Use descriptive and succinct names for all objects
- Good: Underscores ("_"), periods ("."), or combo of upper and lower case
- Avoid: Uninformative names
- Fail: Spaces and certain symbols such as "/"



Good name	Good alternative	Avoid or fail
Max_temp_C	MaxTemp	Maximum Temp (°C)
Precipitation_mm	Precipitation	precmm
Mean_year_growth	MeanYearGrowth	Mean growth/year
sex	sex	M/F
weight	weight	w.
cell_type	CellType	Cell type
Observation_01	first_observation	1st Obs.

3. Straightforward and succint code

- Nobody want to read messy code, including your future self
- Would you want to look at this?

```r
# Not-so-good
for (i in 1:nrow(df <- data.frame(x=c(1, 2, 3, 4)))) { print(df[i,])}
  lst = list(data.frame(y=c(6, 7, 8, 9, 10), z = c(2,2,2,2,2)),
             m = matrix(5,5, nrow=5))

# Better
df <- data.frame(x = c(1:4))
for (i in 1:nrow(df)) {
  print(df[i,])
}
df2 <- data.frame(y = c(6, 7, 8, 9, 10), z = c(rep(2, 5)))
mat <- matrix(5, 5, nrow = 5)
lst <- list(df2, mat)
```

3. Straightforward and succint code

- Use multiple lines to create new objects, etc.
- Reduce text using operators and functions (e.g., `:` and `rep()`)

```r
# Not-so-good
for (i in 1:nrow(df <- data.frame(x=c(1,2,3,4)))) { print(df[i,])}
  lst = list(data.frame(y=c(6,7,8,9,10), z = c(2,2,2,2,2)),
                   m = matrix(5,5, nrow=5))

# Better
df <- data.frame(x = c(1:4))
for (i in 1:nrow(df)) {
  print(df[i,])
}
df2 <- data.frame(y = c(6:10), z = c(rep(2, 5)))
mat <- matrix(5, 5, nrow = 5)
lst <- list(df2, mat)
```

4. Limit the length of a line of code

- Again, short lines of code are easier to read and understand

```r
# Not-so-good
for (i in 1:nrow(df <- data.frame(x=c(1,2,3,4)))) { print(df[i,])}
  lst = list(data.frame(y=c(6,7,8,9,10), z = c(2,2,2,2,2)),
             m = matrix(5,5, nrow=5))

# Better
df <- data.frame(x = c(1:4))
for (i in 1:nrow(df)) {
  print(df[i,])
}
df2 <- data.frame(y = c(6:10), z = c(rep(2, 5)))
mat <- matrix(5, 5, nrow = 5)
lst <- list(df2, mat)
```

5. Use comments frequently

- Allows for explanations, instructions, etc. for analyses
- Your future self and others who review your code will thank you
- Especially important if code is in a script (rather than R Markdown)

```r
# This is a script for degree-day modeling
# Author: Brittany Barker
# Last updated: Jan 13, 2024
library(here) # Useful for project-relevant paths

# Lower developmental threshold (base 50F)
LDT50 <- 50

# Create an empty data frame to store results
out_all <- data.frame(matrix(ncol = 4, nrow = 0))

# Etc, etc.
```

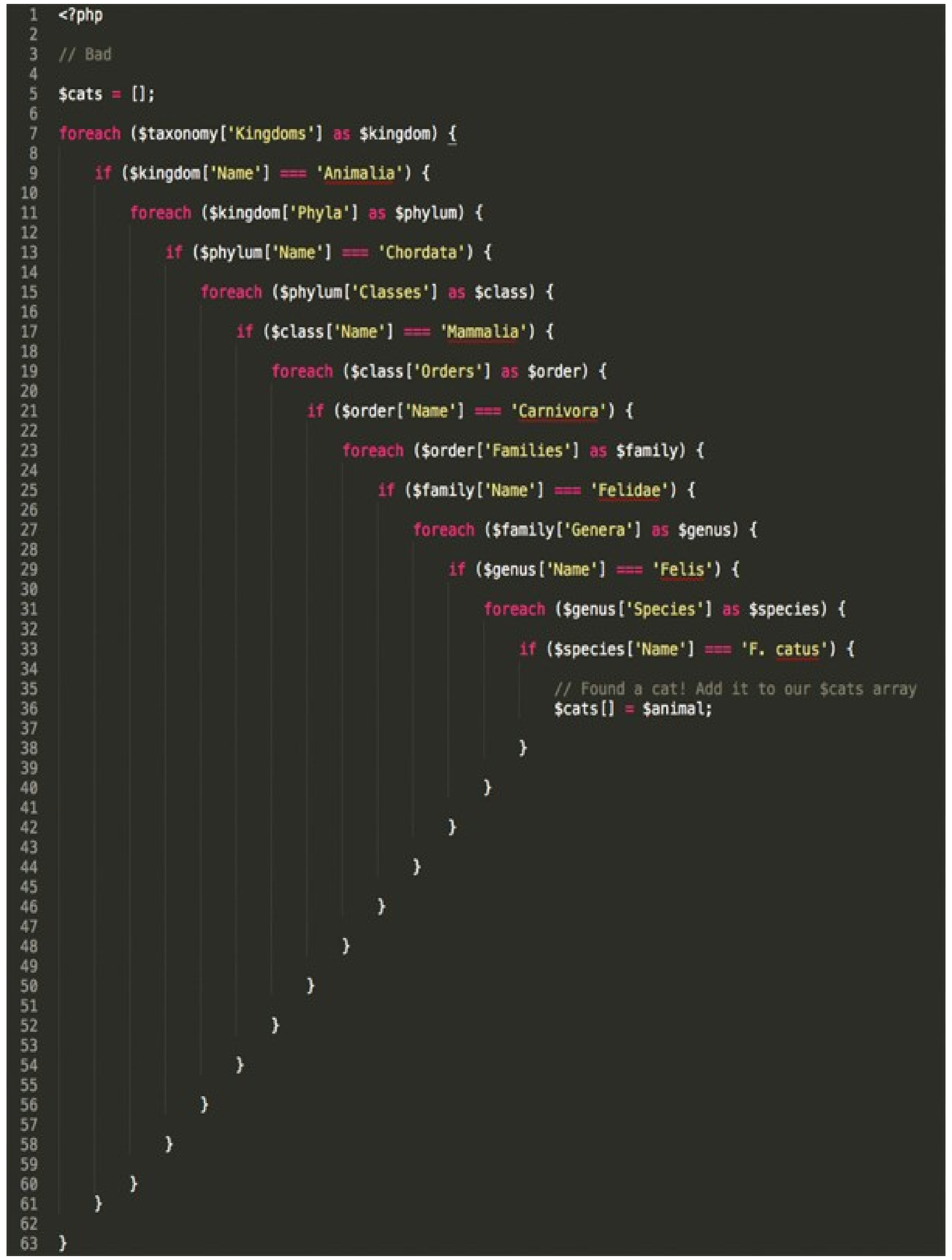
6. Use consistent indentation

- Use tabs, not spaces
- Helps keep track of nested code blocks
- RStudio will auto-indent!

```r
# Notice the 4 levels of indentation
years <- 2022:2023
days <- 1:3
# Loop through each year in `years`
for (year in years) {
  if (year %% 2 == 0) {
    # Next loop through each day
    for (day in days) {
      msg <- paste("Day", day, "in", year)
      print(msg)
    }
  }
}
```

7. Avoid deep nesting

- Keep control structures short
- Easier to trouble-shoot when code breaks
- Increases readability and helps avoid unwanted results
- Pyramid Of Doom: an unwieldy number of nested conditional statements or functions



```php
<?php

// Bad

$cats = [];

foreach ($taxonomy['Kingdoms'] as $kingdom) {

    if ($kingdom['Name'] === 'Animalia') {

        foreach ($kingdom['Phyla'] as $phylum) {

            if ($phylum['Name'] === 'Chordata') {

                foreach ($phylum['Classes'] as $class) {

                    if ($class['Name'] === 'Mammalia') {

                        foreach ($class['Orders'] as $order) {

                            if ($order['Name'] === 'Carnivora') {

                                foreach ($order['Families'] as $family) {

                                    if ($family['Name'] === 'Felidae') {

                                        foreach ($family['Genera'] as $genus) {

                                            if ($genus['Name'] === 'Felis') {

                                                foreach ($genus['Species'] as $species) {

                                                    if ($species['Name'] === 'F. catus') {

                                                        // Found a cat! Add it to our $cats array
                                                        $cats[] = $animal;

                                                    }

                                                }

                                            }

                                        }

                                    }

                                }

                            }

                        }

                    }

                }

            }

        }
    }

}
```