CMSC201 Computer Science I for Majors

Lecture 18 – String Formatting

Last Class We Covered

- Recursion
  - Recursion
    - Recursion
- Fibonacci Sequences
- Recursion vs Iteration

Any Questions from Last Time?

Today’s Objectives

- To understand the purpose of string formatting
- To examine examples of string formatting
  - To learn the different type specifiers
- To briefly discuss tuples
- To learn the details of string formatting
  - Alignment
  - Fill characters

Basic String Formatting

Common Use Cases

- How can we…
  - Print a float without the decimals?

    ```
    print( int(myFloat) )
    ```

    - But what if we wanted it rounded up?
  - Line information up into columns?

    ```
    print(column1, "\t", column2)
    ```

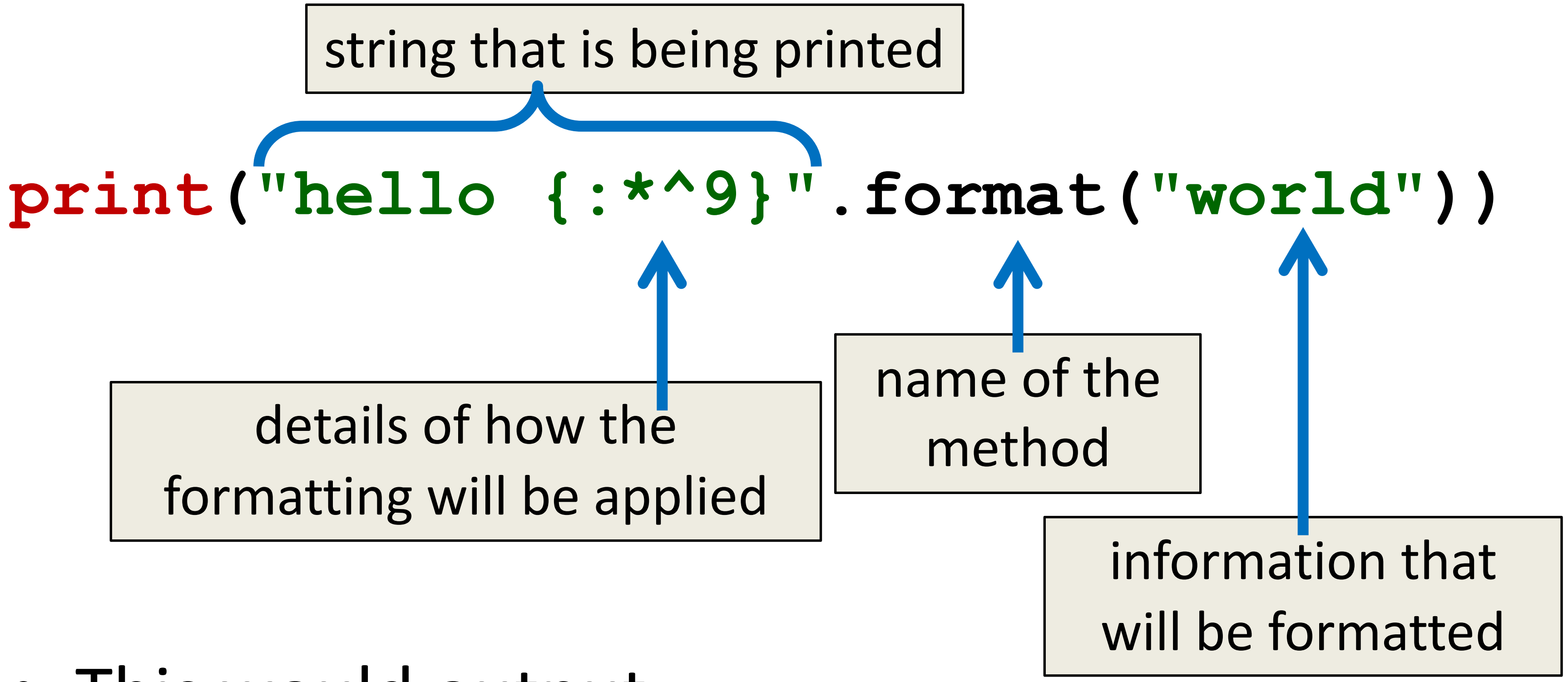
    - But what about when one thing is very long/short?

Accomplishing either of these would require a lot of extra work

String Formatting Possibilities

- Align text left, right, or center
- Create “padding” around information
- Choose the padding character
- Control precision of floats
  - Including automatically rounding up

Anatomy of String Formatting



```
print("hello {:*^9}".format("world"))
```

string that is being printed
details of how the formatting will be applied
name of the method
information that will be formatted

- This would output:

```
hello **world**
```

Type Specifiers

- String formatting often needs to know the exact <u>type</u> of the data it's formatting
  - Or at least how it should be handled

- The three specifiers are

`d`	integer
`f`	float
`s`	string

These are common specifiers shared by many languages, including Python, C/C++, and Java.

Integer Formatting Examples

```
>>> classNum = 201
>>> print("Welcome to {}!".format(classNum))
Welcome to 201!
```

If nothing is specified, no formatting is applied

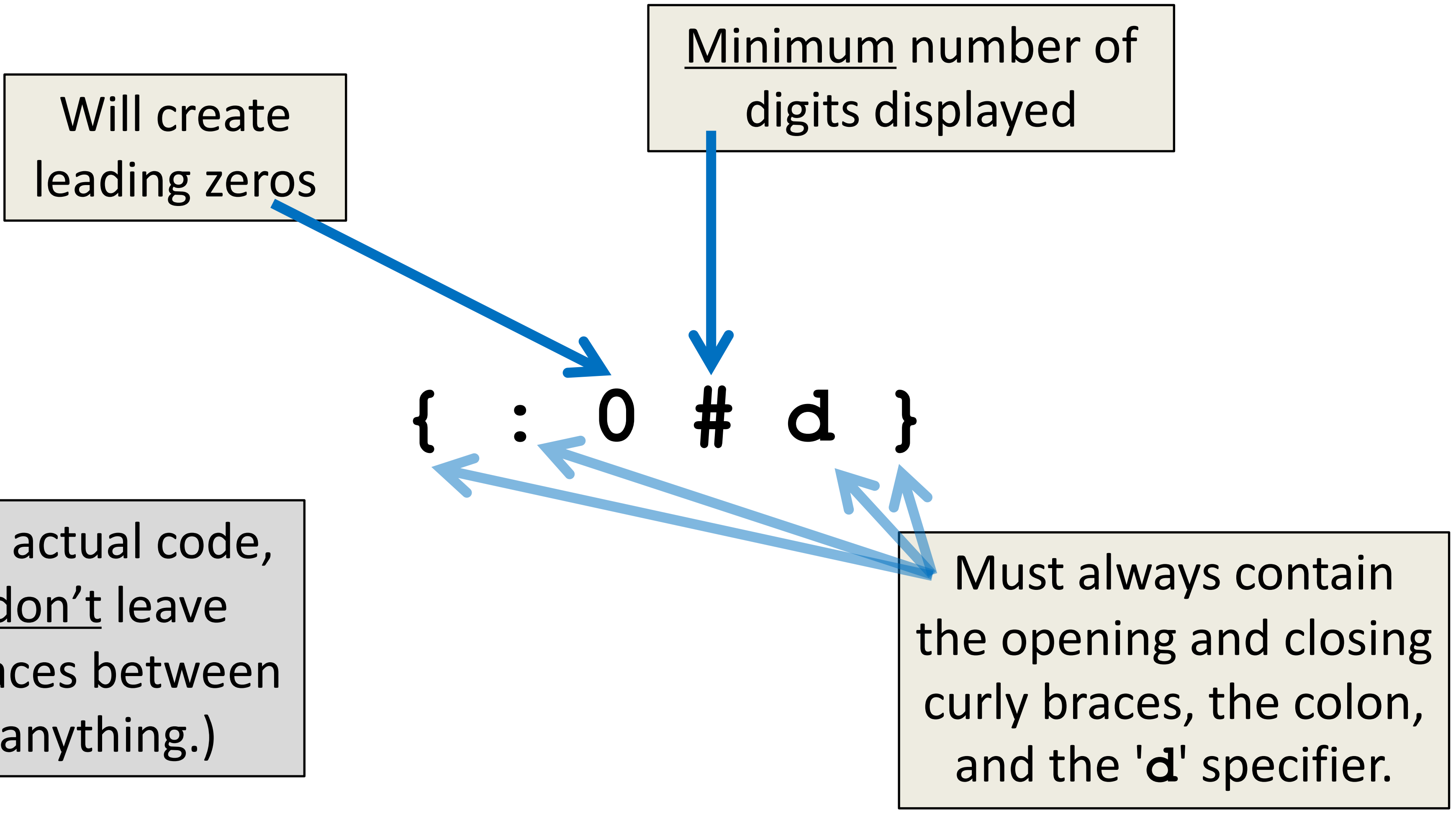
```
>>> print("Welcome to {:5d}!".format(classNum))
Welcome to   201!
```

Specifying “too many” digits will add padding

```
>>> print("Welcome to {:05d}!".format(classNum))
Welcome to 00201!
```

Adding a zero in front will make the padding be zeros

Integer Formatting "Rules"



Will create leading zeros

<u>Minimum</u> number of digits displayed

```
{ : 0 # d }
```

Must always contain the opening and closing curly braces, the colon, and the '`d`' specifier.

(In actual code, <u>don't</u> leave spaces between anything.)

Float Formatting Examples

```
>>> midAvg = 142.86581
>>> print("The midterm average was {:2.0}".format(midAvg))
The midterm average was 1e+02
>>> print("The midterm average was {:2.0f}".format(midAvg))
The midterm average was 143
```

Need to specify that it's a float to prevent truncation

```
>>> print("The midterm average was {:3.1f}".format(midAvg))
The midterm average was 142.9
>>> print("The midterm average was {:1.3f}".format(midAvg))
The midterm average was 142.866
```

Floats will never "lose" the numbers before the decimal

Float Formatting Examples

```
>>> midAvg = 142.86581
>>> print("The midterm average was {:15f}".format(midAvg))
The midterm average was      142.865810
```

Specifying "too many" digits will add padding

```
>>> print("The midterm average was {:015f}".format(midAvg))
The midterm average was 00000142.865810
```

Adding a zero in front will make the padding be zeros

```
>>> print("The midterm average was {:.9f}".format(midAvg))
The midterm average was 142.865810000
```

"Too many" digits after the period will add trailing zeros to the decimal (never spaces)

Float Formatting “Rules”

Will create leading zeros

<u>Minimum</u> number of *total* characters displayed (including " . ")

```
{ : 0 # . # f }
```

(In actual code, <u>don’t</u> leave spaces between anything.)

<u>Maximum</u> number of digits after decimal

Will automatically round, or will pad with trailing zeros

String Formatting Examples

```
>>> best = "dogs"
>>> print("{} are the best animal".format(best))
dogs are the best animal
```

If nothing is specified, no formatting is applied

```
>>> print("{:7s} are the best animal".format(best))
dogs    are the best animal
```

Specifying “too many” characters will add padding

```
>>> print("{:07s} are the best animal".format(best))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: '=' alignment not allowed
            in string format specifier
```

Doesn’t work with strings! (At least, not by itself.)

String Formatting “Rules”

Minimum number of characters displayed

```
{  :  #  s  }
```

(In actual code, don’t leave spaces between anything.)

String Formatting on Multiple Items

Applying to Multiple Items

- To apply string formatting to more than one variable (or literal) within a string, simply use
  - Two sets of **{}** braces with formatting info
  - Two items in the parentheses at the end

```
>>> major = "CMSC"
>>> print("Ready for {:10s} {:04d}?".format(major, 202))
Ready for CMSC       0202?
```

- Will be matched up based on their order

Possible Multiple Item Errors

- If there are too many items
  - Python ignores the extra ones at the end

```
>>> print("It's {:10s} {:2d}, {:4d}".format("April", 16, 2018, "MD"))
It's April      16, 2018
```

- If there are too many sets of **{ }** braces
  - Python will throw an error

```
>>> print("It's {:10s} {:2d}, {:4d}".format("April", 16))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
```

The what index?

Quick Side Note: Tuples

- Tuples are a data structure nearly identical in behavior to lists
  - Lists use square brackets [ ]
  - Tuples use parentheses ( )
- Tuples are *<u>im</u>*mutable
  - Can be indexed, sliced, concatenated, etc.
  - Does not allow “in place” editing or appending

Getting Fancy

Alignment Options

- Can left, right, or center align with formatting:
  - Left <
  - Right >
  - Center ^

In Python 3, left is the default for strings, and right is default for numbers

```
>>> print("why not {:6s}?".format("both"))  # default
why not both  ?
>>> print("why not {:>6s}?".format("both")) # right
why not   both?
>>> print("why not {:^6s}?".format("both")) # center
why not  both ?
```

Padding Characters

- Default padding for strings is spaces
- Default padding for numbers is zeros

- Can replace padding with any single character
  - To prevent errors, specify the alignment too

```
>>> print("why not {:+<6s}?".format("both"))
why not both++?
>>> print("Is this {:~^8d}?".format(currYear))
Is this ~~2018~~?
```

Using Variables

- You can use variables for any of the values in the formatting (size, padding character, etc.)
  – Must use concatenation to put together

```
>>> c = "~"
>>> print( ("why not {:" + c + "^7d}?").format(2))
why not ~~~2~~~?
```

- A better way is to make the string first

```
>>> sentence = "why not {:" + c + "^7d}?"
>>> print(sentence.format(2))
```

"Rules" for Fancy Stuff

Padding character comes right after :

Must have an alignment if you have padding character

```
{ : X < otherStuff }
```

(In actual code, don't leave spaces between anything.)

All the other formatting info comes after these two

Example Usage of Formatting

```
kennel = ["Akita", "Boxer", "Collie", "Dalmatian", "Eurasier"]
for i in range(len(kennel)):
    print("There is a {:>10s} in pen".format(kennel[i]), i)
```

– What would the outcome be here?

```
There is a      Akita in pen 0
There is a      Boxer in pen 1
There is a     Collie in pen 2
There is a  Dalmatian in pen 3
There is a   Eurasier in pen 4
```

String Formatting Exercises

Formatting Exercises

```
print("My dog {}.".format("Hrabowski"))
```

- What formatting is needed for each outcome?

```
My dog    Hrabowski.

My dog Hrabowski   .

My dog _Hrabowski_.

My dog _Hrabowski__.
```

Formatting Exercises

```
print("My dog {}.".format("Hrabowski"))
```

- What formatting is needed for each outcome?

```
My dog   Hrabowski.
   {:>11s}
My dog Hrabowski  .
   {:<11s}
My dog _Hrabowski_.
   {:_^11s}
My dog _Hrabowski__.
   {:_^12s}
```

Left aligned is default, so specifying isn't technically necessary. `{:11s}`

If perfect centering isn't possible, the extra character goes on the right.

More Formatting Exercises

```
PI = 3.14159265358979323846264433
print("Isn't {} great?".format(PI))
```

- What formatting is needed for each outcome?

```
Isn't 3.141593 great?


Isn't   3.141593 great?


Isn't 003.14 great?
```

More Formatting Exercises

```
PI = 3.14159265358979323846264338
print("Isn't {} great?".format(PI))
```

- What formatting is needed for each outcome?

```
Isn't 3.141593 great?
    {:.6f}
Isn't   3.141593 great?
    {:10f}
Isn't 003.14 great?
    {:06.2f}
```

The default is also 6 decimal values.
`{:f}`

Padding numbers with zeros doesn't require an alignment.

Even More Formatting Exercises

- What formatting would be generated here?

```
print("{:1.3f}".format(PI))

print("{:*^10s} is great!".format("Neary"))

print("It's over {:0<4d}!".format(9))

print("{:>7s} {:^^7s}".format("Hello", "world"))
```

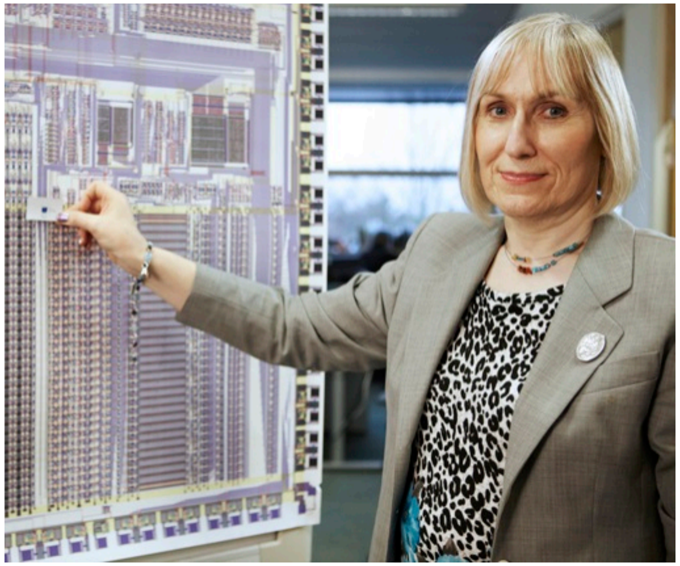
Even More Formatting Exercises

- What formatting would be generated here?

```
print("{:1.3f}".format(PI))
    3.142
print("{:*^10s} is great!".format("Neary"))
    **Neary*** is great!
print("It's over {:0<4d}!".format(9))
    It's over 9000!
print("{:>7s} {:^^7s}".format("Hello", "world"))
   Hello ^world^
```

Daily CS History

- Sophie Wilson
  - Designed the Acorn Micro-Computer in 1979
    - Wrote BBC BASIC, the programming language
  - Designed the instruction set of the ARM processor
    - Most widely-used architecture in modern smartphones



Announcements

- Project 2 is due Friday 11/9 at 8:59:59PM

- Midterm #2 is next week!

Image Sources

- Sophie Wilson (adapted from)
  - https://www.flickr.com/photos/101251639@N02/9669448671