

ITEC 1150 Week 3

Chapter 1 Part 2

FORMATTING, INPUT & MORE ON DATA TYPES

Check in

- ▶ Remember, submit assignments by the due date, even if you're not done. You'll receive feedback from me—an example, video, notes, etc. Then you'll be able to resubmit for regrading.
- ▶ Be sure to pay attention to details in the lab assignments. For the first couple of weeks, not all details are required (for example, the \$ symbol in some of the Week 2 labs), but details like that *will* be graded in the future. Work on observing and fixing.
- ▶ Remember: examples provide a solution – not *the* solution. There are almost always multiple ways to get to a correct solution.

Program Development Plan

4 Program Development Plan for Grade Averaging Program

1. Problem Definition

Write a program to calculate and display the average quiz score of three students.

2. Variables/ calculations/ functions

score1 = 45

score2 = 74

score3 = 63

`score_avg = (score1 + score2 + score3)/3`

→ rounding may be desirable

3. Algorithm - steps in order

1. Save the first student score under a variable name.
2. Save the second score.
3. Save the third score.
4. Calculate the average by adding the 3 scores and dividing the sum by 3.
5. Display the average in a meaningful sentence. Note: may require rounding and/or formatting to look good.

4. Coding

See separate file: avg_grade.py

5. Testing – screen snips plus comments

On first test, the program output had a long decimal and no period at the end of the sentence.

The average test score is 60.666666666666664

Fixed by using the `round()` function against the average score calculation (slide 19), and adding a '.' after the variable in the output string. In order to get the period to add on properly, I had to switch from comma method of string concatenation, to the + sign method (slide 18).

The average test score is 60.67.

6. Improvement – new features needed for next version

This program would be more useful if the teacher could enter scores, rather than hard coding scores. Also, you could collect "points possible" as a separate variable, so the % correct could be calculated. More data analysis could be provided like min and max. It would be better if you asked for the number of students in the class, and then you could ask for scores for each one.

► Do you ever think about ways that apps you use could be better?

► In fact, you use the program development cycle all the time, when you observe what could be improved in your apps. Programs evolve or die out!

← In D2L resources, find a Word template to use for the PDP, with 6 sections.

► Even though you don't have to submit a PDP, always use the thought process.

Program Model – under construction

- ▶ When we adapt a model, we DON'T change things that are working well.
- ▶ Instead, we change details & add features.
- ▶ In this lecture, we will adapt the `bus_fare.py` program from week 2 to produce the info in the table on the right →
- ▶ It will be named `bus_fare2.py`
- ▶ After producing the data points, we will study new formatting tricks to print in columns.
- ▶ You will need to adapt models, and format data in sentences and tables for the rest of the term, so this is an important lesson!

The monthly bus fare for 19 trips = \$48.25

Type	Trips	Fare	Total
Regular	7	\$ 1.75	\$ 12.25
Rush hour	12	\$ 3.00	\$ 36.00

String Formatting

CONCATENATION AND THE FORM OF THE `PRINT()` FUNCTION THAT TAKES MULTIPLE VALUES ARE BOTH FINE FOR SIMPLE CASES BUT BECOME ERROR-PRONE AS OUR STRINGS GET MORE COMPLEX.

STRINGS THEMSELVES HAVE FUNCTIONS WHICH ALLOW US MUCH BETTER CONTROL OVER OUR OUTPUT.

The String Format Method

- ▶ Try this in your lesson1-2exercise.py file:
 1. `user_name = 'Erik'`
 2. `output = 'Hello {}, it\'s nice to meet you!'.format (user_name)`
 3. `print(output)`
- ▶ Line 2 uses the **format** method on the string. Notice that the string is immediately followed by `".format()"`. That's the Python way of saying, "Format this string for me."
- ▶ We pass `user_name` to the `format()` method, and Python will replace the braces `{}` with whatever is in the variable.

The word "method" can be confusing. Here, it doesn't mean "process" but refers to a function that is part of another thing.

`string.format()` ← method
`print()` ← plain function

You don't need to memorize this for now; just an explanation if the use of "method" was confusing.

The String Format Method (cont.)

- ▶ We can use as many variables as we like, and they will be matched in order with sets of braces. ():

```
1. user_name = 'Erik'
2. my_name = 'Alice'
3. output = 'Hello {}, it\'s nice to meet you! My name is {}'.format (user_name, my_name)
4. print(output)
```

- ▶ You will get an error if you pass fewer variables than you have pairs of braces:

```
1. output = 'Hello {}, it\'s nice to meet you! My name is {}'.format (user_name)
```

- ▶ If you pass *more* variables than you have pairs of braces, the extra variables will be ignored. This can be a source of bugs!

Two sets of braces, but only one variable causes an error!

The String Format Shortcut Method

- ▶ Try this in your lesson1-2exercise.py file:

```
1. user_name = 'Erik'
2. my_name = 'Alice'
3. output = f'Hello {user_name}, it\'s nice to meet you! My name is {my_name}.'
4. print(output)
```

- ▶ The shortcut method uses an 'f' right before your string.
- ▶ The variable names are then put inside the braces.
- ▶ Using either style of the Format method is acceptable—use which ever one works best for you.

You may also run across this old-school formatting style online; it is no longer used in modern Python:

```
print('Hello %s, it\'s nice to meet you!' %
user_name) # Python 2.x method
```


Formatting Codes

WE'VE PUT VARIABLES INTO
STRINGS BUT HAVEN'T
CHANGED THEIR FORMAT
YET.

WE DO SO WITH
FORMATTING CODES.

Formatting Codes Example

- There are many options. Here's a 2-column table for your exercise file.

```
# two column table using a "leader" character
city = 'Minneapolis'
temp = 67
chance_precip = 40
parking = 20.00
guests = 1000

print(f'Welcome!\n{city} Hilton Today')
print('{:.<18}{1:>5d}f'.format('Temperature', temp))
print('{:.<18}${: >5.2f}'.format('Parking', parking))
print('{:.<18}{: >5d}%'.format('Chance Precip', chance_precip))
print('{:.<18} {: >5,d}'.format('Guests', guests))
```

- Here's the output:

```
Welcome!
Minneapolis Hilton Today
Temperature.....67f
Parking.....$20.00
Chance Precip.....40%
Guests..... 1,000
```

← We'll look at what those codes mean in the next slides.

Format Codes


- ▶ **format_spec:** [fill][align][width][grouping_option]["." precision][type]
- ▶ This is a (partial) *specification* for formatting codes in Python.

Controls	Description	Allowed Values
Fill	If you're putting space around the value, you can choose to replace the empty spaces. Only works if Width is specified.	Any character
Align	If you're putting space around the value, choose to left align, right align, or center the value. Only works if Width is specified.	< Left > Right ^ Center
Width	How wide do you want the column around the value? This adds spaces to the length of the value including the value to equal the width. If the value is longer than the width, your columns will not be aligned!	Any positive integer
Grouping Option	Thousands separator.	, _
Precision	How many decimal places do you want to show? Only works with Float, not Integer.	Any positive integer
Type	Can be used to change the type of the variable for formatting purposes.	Lots, but we'll only use 'f' to make sure our numbers are treated as floats when we want decimal places.

Format Code Syntax: .format() method

- ▶ The formatting code always follows a colon : and the colon always follows the value.
- ▶ In the .format() method on line 2, Python inserts the values in order, so we don't put anything in front of the colon.

```
1. temp = 72  
2. print('{:.<18}{:.>5}F'.format('Temperature', temp))
```



The values passed don't have to be in variables. This is useful for adding headings such as 'Temperature' in the example.

Format Code Syntax: Shortcut Method

- ▶ The formatting code always follows a colon : and the colon always follows the value.
- ▶ With the shortcut method, the values are put inside the braces and go before the colon.

1. `temp = 72`
2. `print(f'{"Temperature":.<18}{temp:.>5}F')`

Format Code Example: Width, Fill, and Alignment

format_spec: [fill][align][width]

1. temp = 72

2. print(f'{"Temperature":.<18}{temp:.>5}F')

- ▶ In the first formatting block, we're specifying:
 - ▶ Fill with periods → .
 - ▶ Align to the left → <
 - ▶ Make the column 18 characters wide → 18
- ▶ In the second formatting block, we're specifying:
 - ▶ Fill with periods → .
 - ▶ Align to the right → >
 - ▶ Make the column 5 characters wide → 5

That formatting results in this output:

Temperature.....72F

If you count the characters, you see that we have 24, which is the 18 + 5 we specified for our column widths, plus one more for the Fahrenheit symbol ("F") which isn't in a formatting block.

Format Code Example: Grouping Option

`format_spec: [fill][align][width][grouping_option]`

1. `salary = 54321`

2. `print(f'{"Salary":.<16}${salary:>7,}')`

The Grouping option is much easier! The comma after the width (7) tells Python to format the value with a comma as the thousands separator:

```
Salary.....$ 54,321
```

Format Code Example: Precision and Type

► **format_spec:** [fill][align][width][grouping_option]["." precision][type]

1. salary = 54321

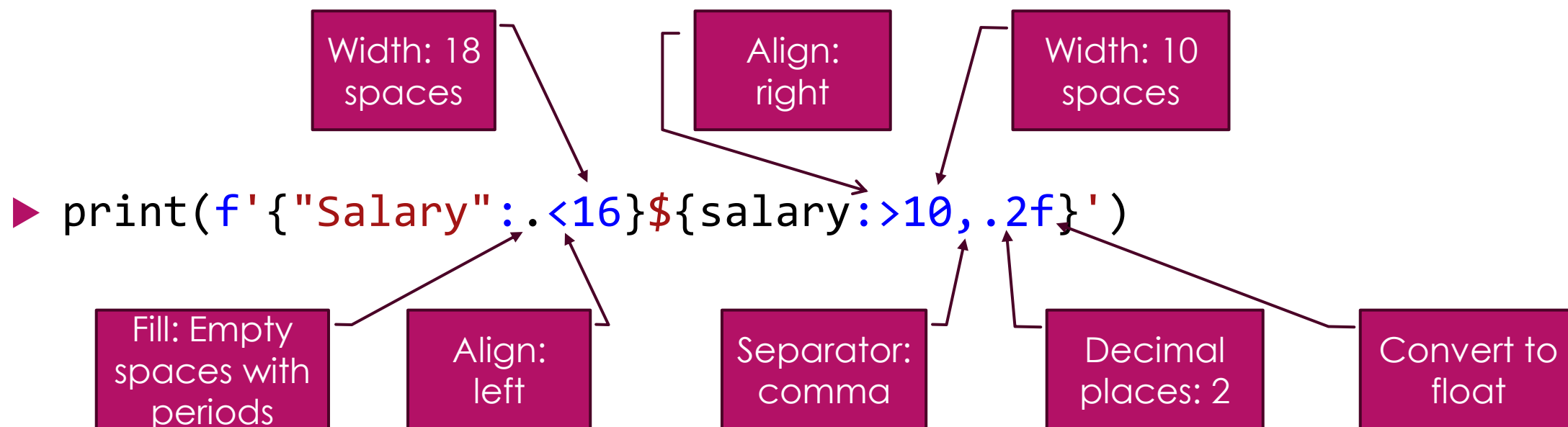
2. print(f'{"Salary":.<16}\$ {salary:>10,.2f}')

- The Precision code always starts with a decimal point (period) and is followed by the number of decimal places you want to show.
- The Type code is used to tell Python to convert the value to the type specified. If you want to show decimal places, always use 'f' as the type code so that if you pass in an integer, Python will convert it to a float.
- Our example uses an integer, but by specifying .2f, it gets treated as a float and two decimal places are shown:

```
Salary.....$ 54,321.00
```


Complete Formatting Example

► `[fill][align][width][grouping_option][." precision][type]`



Formatting Model for Columns

Example: in the bus_fare2.py program, we needed to print info in columns.

Try to explain the code below, which produces the output shown on slide 4.

1. `print(f'Total monthly bus fare for {total_rides} trips = ${total_cost:,.2f}.')`
2. `print('{:<15}{:>5}{:>6}{:>5}{:>6}{:>6}'.format('Type', 'Trips', '', 'Fares', '', 'Total'))`
3. `print('{:<15}{:>5}{:>6}{:>5.2f}{:>6}{:>6.2f}'.format('Regular', regular_rides, '$', regular_fare, '$', regular_total))`
4. `print('{:<15}{:>5}{:>6}{:>5.2f}{:>6}{:>6.2f}'.format('Rush', rush_hour_rides, '$', rush_hour_fare, '$', rush_hour_total))`

Tip: Column Layout

- ▶ If you're having trouble with figuring out the width of columns, laying them out in a tool like Excel can help. In the image below, I made a bunch of narrow cells in Excel and put one character in each one.
- ▶ When it looked right, I could count how many characters I needed for each column.

[illegible]

User Input

MAKING OUR PROGRAMS
DYNAMIC!

Hello World, But Better

- ▶ Instead of hard-coding data, we can have Python ask the user for input.
- ▶ We can store that input in a variable, and use it elsewhere
- ▶ Here's some code that changes Hello World to say hello to the user instead. It asks for a name within the `input()` function, and then prints a greeting message.

```
1. name1 = input('Please type your name here >>> ')\n2. print('Hello, ' + name1 + '!')
```

The string inside the input function is displayed to the user.

Album Count Revisited

1. `number_of_cds = 45`
2. `number_of_lps = 23`
3. `number_of_albums = number_of_cds + number_of_lps`
4. `print('With', str(number_of_cds), 'CDs and', str(number_of_lps), 'LPs, the number of albums you have is', str(number_of_albums)+ '.')`

- ▶ It's a great program if you have 45 CDs and 23 LPs.
- ▶ Programs will be better, e.g., dynamic, if the users can tell the program how many CDs and LPs they have. We will adapt this program, but first...

Getting Numeric Data – A Wrinkle

- ▶ Python's `input()` function only accepts string data
- ▶ No matter what you enter, it will be stored as a string type.
- ▶ If you need numerical data, tell the program to store the user's input in an integer variable or a floating-point variable
 1. `number_of_cds = int(input('How many CDs do you own?'))`
 2. `age = float(input('Enter your age - use decimal for partial year.'))`

Note: if the user enters "abc" as the input, this will cause an error as Python can't convert "abc" to a number. We will learn how to guard against that problem in the next lesson.

Integer Conversion of Input Data

```
1. number_of_cds = int(input('How many CDs do you have? '))
2. number_of_lps = int(input('How many LPs do you have? '))
3. number_of_albums = number_of_cds + number_of_lps
4. print('With', number_of_cds, 'CDs and', number_of_lps, 'LPs, the
    number of albums you have is', str(number_of_albums) + '.')
```

- ▶ Note the `int()` and `str()` conversions – the data type is going back and forth as needed.
- ▶ Note nesting—Use correct order/ position to nest the functions

Floating Point Examples

- ▶ Like integers, for example:

1. `temp = float(input('Enter the patient\'s temperature: '))`
2. `pay = float(input('Enter the employee\'s hourly pay: '))`
3. `print('Your temp is', temp, 'degrees and your pay is $' + str(pay))`

Note again:

- ▶ Nesting of functions
- ▶ The use of the escape character to handle an apostrophe
- ▶ The use of , vs + to combine pieces of a print statement (joining by comma automatically includes a space).

Flip flopping Data types

You often convert user input to a numeric data type, to do math. Then you must sometimes convert results back to string data for printing. Formatting methods can make things easier. This code will be part of a lab, so put it in a new file called `coffee_sales.py`

```
1. cups = int(input('How many cups of coffee sold? '))
2. price = float(input('Price of coffee? '))
3. coffee_sales = cups * price
4. print(f'Total sales = ${coffee_sales:,.2f}')
```

- ▶ Run the code with test data: 40 cups of coffee sold @ price 2.50. Now try 500 cups @ 2.25. To throw an error, try saying you sold 1.5 cups of coffee.
- ▶ What type of variable is `coffee_sales` automatically assigned?
- ▶ Practice: now let's make a 2-column table to display the details.

Formatting Mania

- Showing details...add to your code from the previous slide, run & analyze.

```
print(f'Gross sales today, for {cups} cups of coffee: ')\nprint('{:<20}{:>3}{:>8,.2f}'.format('Price per cup', '$', price))\nprint('{:<20}{:>3}{:>8,.2f}'.format('Total sales', '$', coffee_sales))
```

- You can use the code on this slide and the previous slide as a starting point for one of the labs.