Chapter 2

CITC 1301 - A70

D. Blair - Northeast State Community College

The Software Development Process

Computers are literal beasts. They must be told exactly what to do, done to the smallest detail. For that reason, we need to following a systematic approach.

- Analyze the problem (Problem Statement) - yes, all software programs solve a problem and it is up to you to determine, in detail, what the problem is; and, create a detailed problem statement.
- Requirements - Describe exactly what your program will do, not how, but what it will do.
- Specifications - Describe how your program will solve the problem
- Implement the design
- Test the implementation
- Repeat until done

Example:

- Problem Statement - A temperature is given in Celsius and the user wants it to be expressed in degrees Fahrenheit
- Requirement - Input will be entered in Celsius, output will be in Fahrenheit, processing: 9/5 * C + 32
- Specification
    - Prompt user for Celsius
    - Get input from user and put in variable named celsius
    - Run the calculation and put results in variable named fahrenheit
        - fahrenheit = (9 / 5 * celsius) + 32
    - Output the results to the console screen using a print statement
- Test
    - Enter 0 results = 32F
    - Enter 100 results = 212F

Expressions

The fragments of program code that produce or calculate new data values are called expressions. The process of turning an expression into an underlying data type is called evaluation. Variables can hold the evaluation of an expression. A variable has a name associated with it that is used instead of the memory location. For example:

```
firstName = “Foobar”
x = 5
y = 10
```

fahrenheit = 9 / 5 * celsius + 32

Note that spaces are irrelevant. The above formula could just as well have been:

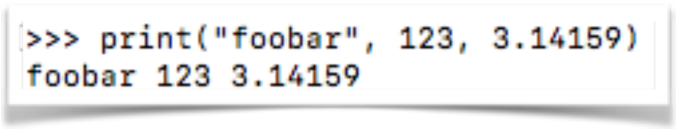
fahrenheit=9/5*celsius+32

But, good coding style tells us to put spaces in to make our code more readable.

Print Function

You will see the print function referred to as the "**print statement**" but in actuality, in Python 3 the print statement was changed to a print function giving it more power.

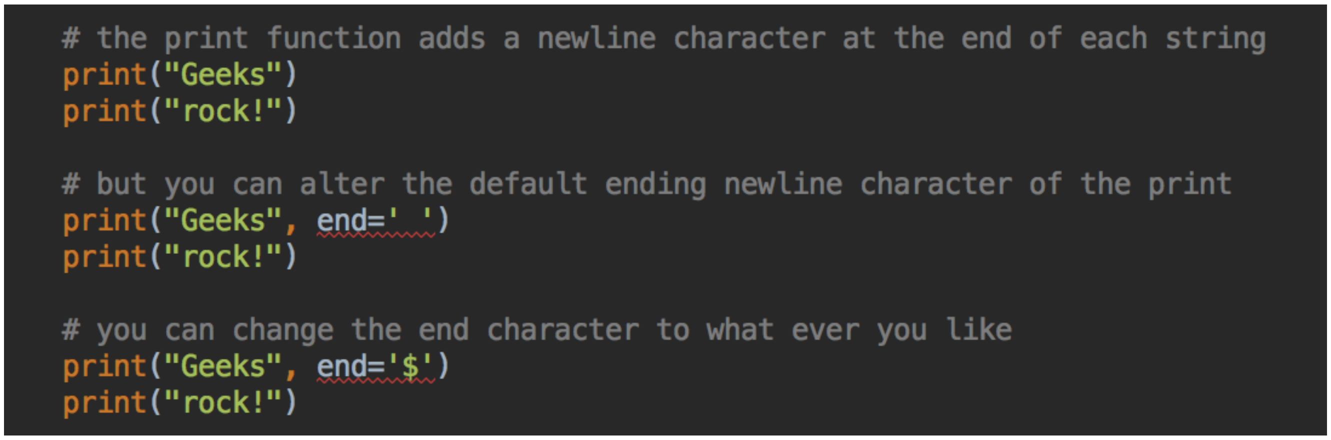
Many times you will want to format your output to be a little fancier than just a list of characters or values separated by a space.



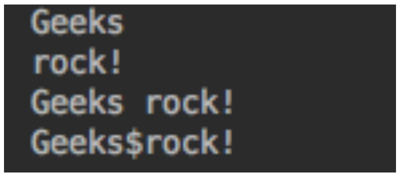
```
>>> print("foobar", 123, 3.14159)
foobar 123 3.14159
```

In the above code, the comma-separated list in the print function parameter list will all be evaluated by print separately and then converted to an output string separated by a single space between each parameter; then concatenated together to form one long string.

If you play around with the print function enough you will soon figure out that it automatically adds a newline character after each print. The way to keep that from happening is to use one of the "overloaded" print functions. You can redefine that the print statement does at the end of the line by redefining **end=** like this:

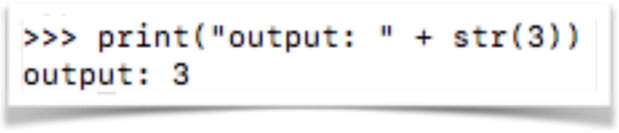


```
# the print function adds a newline character at the end of each string
print("Geeks")
print("rock!")

# but you can alter the default ending newline character of the print
print("Geeks", end=' ')
print("rock!")

# you can change the end character to what ever you like
print("Geeks", end='$')
print("rock!")
```



```
Geeks
rock!
Geeks rock!
Geeks$rock!
```

In the code below, we see that we can use "concatenation" to manually output a single string to the output window BUT if there is a numeric value, it must be manually converted to a string by using the str() function to cast that value to a string. Try leaving the str() out.



```
>>> print("output: " + str(3))
output: 3
```

But, once again, what if we need a "better formatted" output?

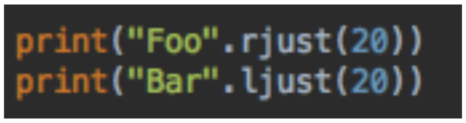
The string class has many useful functions that will help us format the output. We will look at 3 of those right now; ljust(), rjust(), and format().

ljust() and rjust()

ljust() and rjust() stand for left-justified and right-justified, are very easy to use. First, what is a string? A string is a "group" of characters. Usually you can think of a string as something like this: "Hello Mr. Foobar" … but can also look like this: "100 miles". So a number can be a string just like alpha characters, the decimal point, hyphens, special characters, and so on. I can use a variable to represent any of those string combinations that I like.

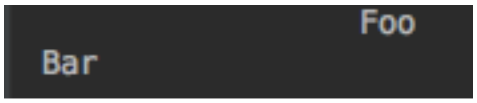
firstName = "Foo"
lastName = "Bar"
Address = "100 N. Main St."

We could have also used input from a user to put into a string. We will see how to do that in just a moment.



```
print("Foo".rjust(20))
print("Bar".ljust(20))
```

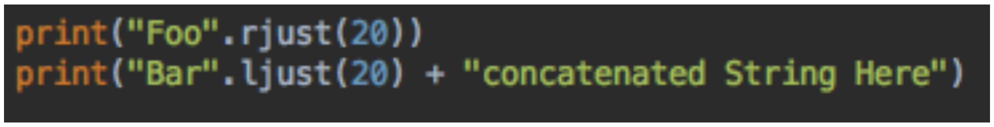
output:



```
                 Foo
Bar
```

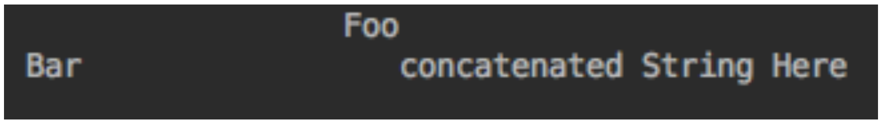
Notice how “Foo” is padded with spaces on the left. rjust(20) printed 20 spaces before printing “Foo” to the console window. Similar for ljust(20) but you don’t notice the spaces to the right of “Bar” because nothing comes after the last space.

We can see that ljust(20) works by concatenating a string to the end.



```python
print("Foo".rjust(20))
print("Bar".ljust(20) + "concatenated String Here")
```

Output:

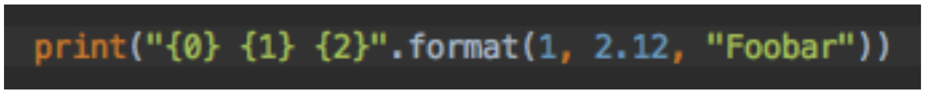


```
                 Foo
Bar                 concatenated String Here
```

Format()

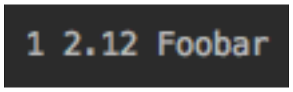
The format() function is a little more difficult to use but has a lot of power. We will look at a small part of what it can do right now.

Let’s take a look at a simple example:



```python
print("{0} {1} {2}".format(1, 2.12, "Foobar"))
```

Output:

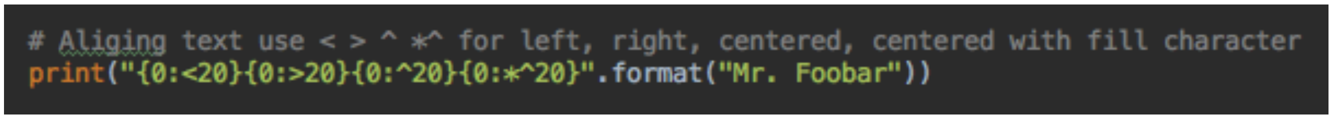


```
1 2.12 Foobar
```

Notice that in the print function, there are three open-close curly braces all in a string defined by the double quotes. Those curly braces are special in that they are replaced by the values in the format function.

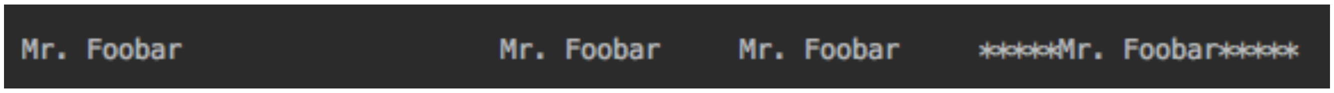
Also, between the string and the format function there is a dot. This dot resolves the string so Python can find its format function that is has contained in the string class. More on this much later. But for now, just know you have to use the dot operator to get to the format( ) function just like you did to get to the ljust( ) and rjust( ) functions.

A more robust example:



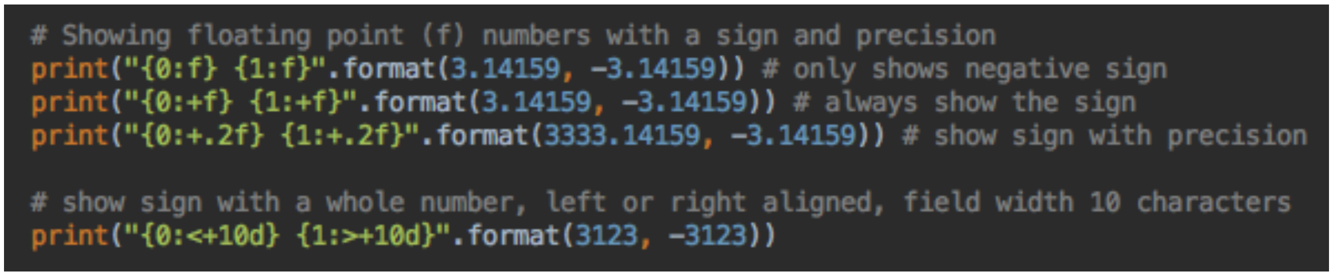
```
# Aliging text use < > ^ *^ for left, right, centered, centered with fill character
print("{0:<20}{0:>20}{0:^20}{0:*^20}".format("Mr. Foobar"))
```

Output

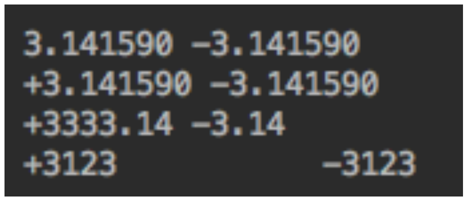


```
Mr. Foobar                    Mr. Foobar     Mr. Foobar     *****Mr. Foobar*****
```

Specifying a sign and using floating point numbers and more:



```
# Showing floating point (f) numbers with a sign and precision
print("{0:f} {1:f}".format(3.14159, -3.14159)) # only shows negative sign
print("{0:+f} {1:+f}".format(3.14159, -3.14159)) # always show the sign
print("{0:+.2f} {1:+.2f}".format(3333.14159, -3.14159)) # show sign with precision

# show sign with a whole number, left or right aligned, field width 10 characters
print("{0:<+10d} {1:>+10d}".format(3123, -3123))
```



```
3.141590 -3.141590
+3.141590 -3.141590
+3333.14 -3.14
+3123           -3123
```

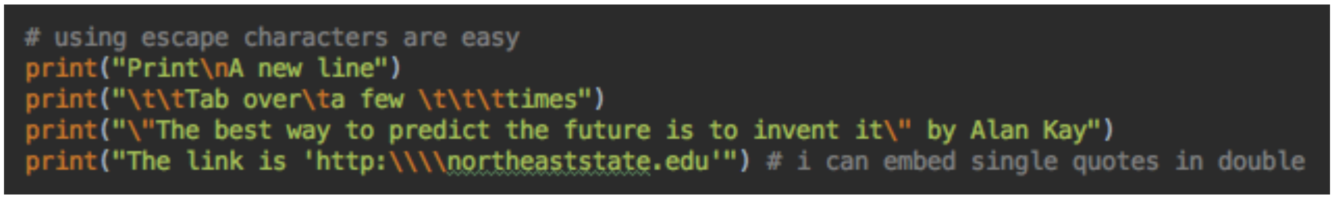
Escape Sequences

An escape sequence within a string literal tells Python to do something special to the output. This is usually add a newline character, add a backslash, add quotes, or tab. But there are other escape sequences but the other are rarely used, and will never be used in this class. In case you wondered, the escape sequences in Python are the same for C, C++, Java, and many other languages.

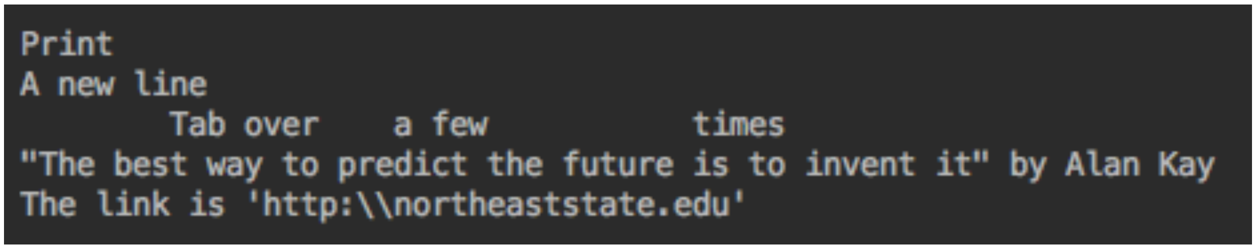
\n = newline character
\t = tab
\\ = backslash

\" = double quotes
\' = single quotes

At first they might be confusing but once you see them in action you will find out they are pretty simple and very useful. Here is how they are used.



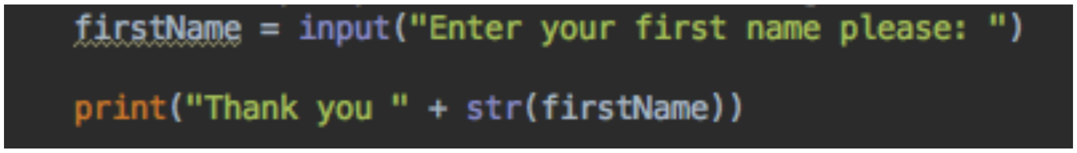
```
# using escape characters are easy
print("Print\nA new line")
print("\t\tTab over\ta few \t\t\ttimes")
print("\"The best way to predict the future is to invent it\" by Alan Kay")
print("The link is 'http:\\\\northeaststate.edu'") # i can embed single quotes in double
```



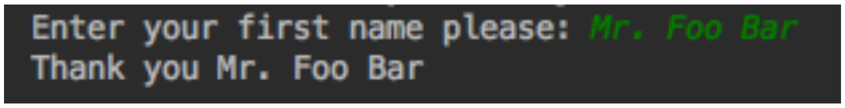
```
Print
A new line
        Tab over    a few           times
"The best way to predict the future is to invent it" by Alan Kay
The link is 'http:\\northeaststate.edu'
```

Getting Input from the User

Getting input from the user is fairly simple. Python has a function called "input( )" that prompts the user with a custom message and return a string that you usually will put into a variable that you create and use later. For example:



```
firstName = input("Enter your first name please: ")

print("Thank you " + str(firstName))
```

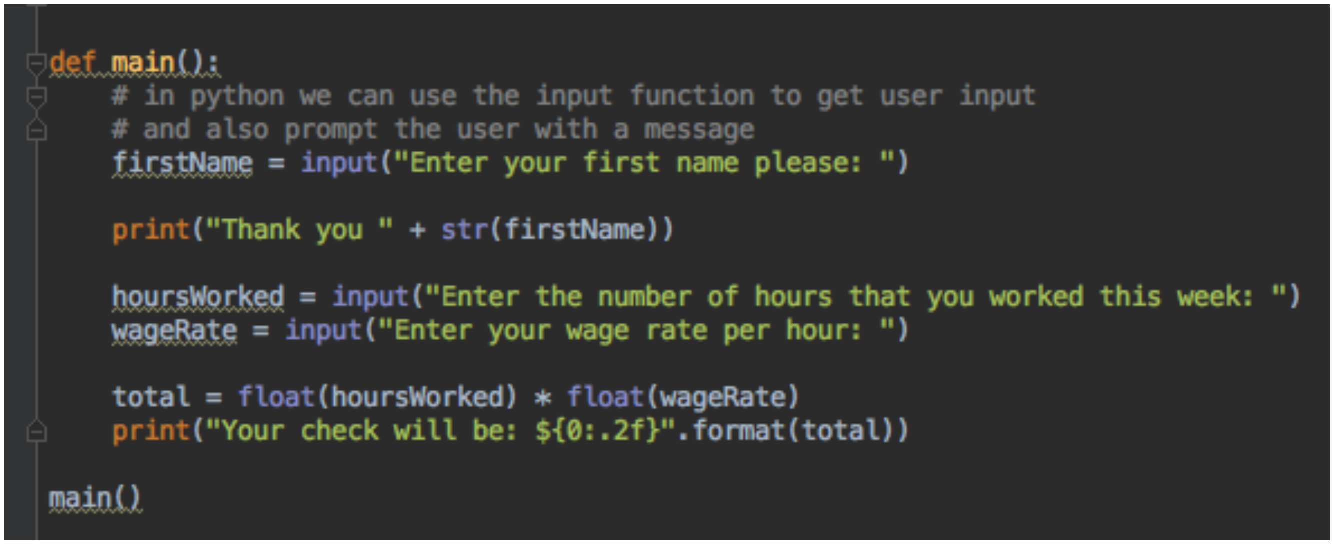


```
Enter your first name please: Mr. Foo Bar
Thank you Mr. Foo Bar
```

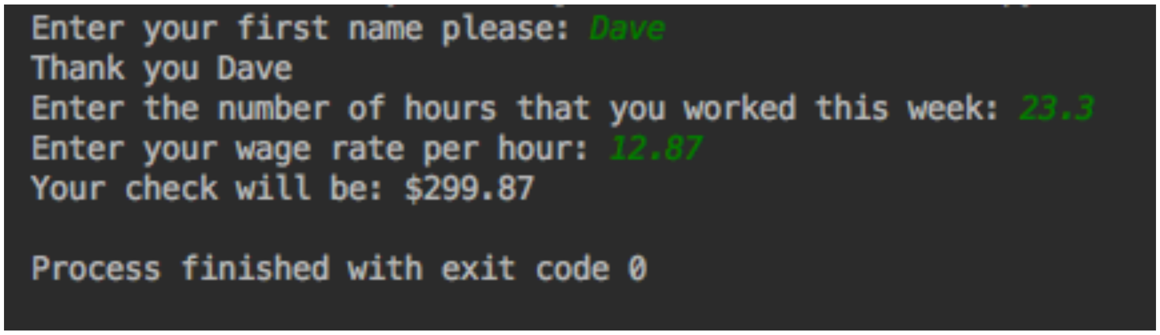
If we want to do any math with the input, then it is required that we convert the string input into an integer of floating point number.

Of course, we will have to see how to validate input a bit later so for now, assume that the user will enter a floating point number for a salary; for example, instead of a word "joe".

Here is a complete example:



```python
def main():
    # in python we can use the input function to get user input
    # and also prompt the user with a message
    firstName = input("Enter your first name please: ")

    print("Thank you " + str(firstName))

    hoursWorked = input("Enter the number of hours that you worked this week: ")
    wageRate = input("Enter your wage rate per hour: ")

    total = float(hoursWorked) * float(wageRate)
    print("Your check will be: ${0:.2f}".format(total))

main()
```



```
Enter your first name please: Dave
Thank you Dave
Enter the number of hours that you worked this week: 23.3
Enter your wage rate per hour: 12.87
Your check will be: $299.87

Process finished with exit code 0
```