CSP1150/CSP5110: Programming Principles
Reading 1.2: A Closer Look at the print() Function

We introduced the "`print()`" function this week. It's pretty simple – it shows stuff on the screen:

```
print('Welcome to Python.')
Welcome to Python.
```
Python

What you print depends on the program, but generally it will either be some kind of information or instructions, or the result of a program. Printing results typically involves printing a variable:

```
# convert pounds to kilograms
value = input('Enter a value in pounds: ')
result = float(value) * 0.454
print(value, 'pounds is', result, 'kilograms.')
```
Python

```
Enter a value in pounds: 5
5 pounds is 2.27 kilograms.
```

In this example, the first line that appears is the prompt specified for the "`input()`" function, and the second line that appears is caused by the "`print()`" function, which has been told to print the variables and bits of text.

In the first example, we gave "`print()`" a single thing to show – "Welcome to Python."
In the second example, we gave it four things to show – the value variable, "pounds is", the result variable and "kilograms." When "`print()`" is given multiple things to show, it converts them all into one big bit of text (a "string") before showing them. It also adds a space between each thing.

```
print(value,      'pounds is',      result,      'kilograms.')
```
Replace variables with their values…
```
    5         'pounds is'         2.27         'kilograms.'
```
Convert any numbers into text (strings)…
```
  '5'         'pounds is'        '2.27'        'kilograms.'
```
Add a space between each part…
```
  '5'   ' '   'pounds is'   ' '   '2.27'   ' '   'kilograms.'
```
Combine it all into one and show it.
```
5 pounds is 2.27 kilograms.
```

Python does all this for you – all you need to do is tell it what to print. But no matter how much a language tries to be convenient, there will always be situations where it isn't. A common example is "What if I don't want a space to be added between everything?"

```
value = int(input('Type a number: '))
print('You typed "', value, '".')
```
Python

```
Type a number: 2
You typed " 2 ".
```

In this example, the output looks bad because a space has been added on either side of the value variable. It would look better as: `You typed a "2".`

There are two options in this situation. One solution is to specify what you want to use between each thing – known as the separator:

```
value = int(input('Type a number: '))
print('You typed "', value, '".', sep='')
```

```
Type a number: 2
You typed "2".
```

Python

In this version, the last thing we've passed to the "`print()`" function is "`sep=''`", which tells it to use a separator of `''` (i.e. nothing, an empty string). If you want a space between some things and not others, you can specify "`sep=''`" and include the spaces in the things themselves as needed.

The second option is to pass everything to the "`print()`" function as one big string, essentially going through the process described on the previous page yourself:

```
value = int(input('Type a number: '))
print('You typed "' + str(value) + '".')
```

```
Type a number: 2
You typed "2".
```

Python

Adding two strings together to form a single string is known as "concatenation", and it is described at the end of Lecture 1. You simply place a "+" between any strings that you want to join together. Since you can only concatenate strings, variables containing numbers must be converted to strings before you concatenate them. This is achieved by putting the "`str()`" function around them.

```
print('You typed "'  +  str(value)  +  '".')
```

Replace variables with their values…

```
'You typed "'  +  str(2)  +  '".'
```

Convert any numbers into text (strings)…

```
'You typed "'  +  '2'  +  '".'
```

Combine it all into one and show it.

```
You typed "2".
```

Which way you use is entirely up to you. The automatic conversion of numbers into strings and automatic adding of spaces between things is useful most of the time, but in some situations you might find it more convenient to convert and concatenate things yourself.

Finally, one other common issue when printing: "How do I print a quote mark or apostrophe?" You can use single or double quote marks around strings in Python (some other languages only allow one or the other), and as long as your string doesn't contain that type of quote mark it will work fine. But if your string *does* contain one then the language gets confused about where the string ends…

```
print('"None shall pass." – The Black Knight') # no problem
"None shall pass." – The Black Knight
print("Don't worry.") # no problem
Don't worry.
print('Don't worry.') # error!
```

Python

The solution is to put a "\" before any quote marks that conflict with the ones used for the string:

```
print('Don\'t worry.') # no problem
Don't worry.
```

Python

The "\" is an "escape character" – it makes the subsequent character be interpreted literally (as just another character), instead of it being interpreted as indicating the end of the string.