1811/2807/7001ICT
Programming Principles

School of Information and Communication Technology
Griffith University

Trimester 1, 2024

5 Programs That Print

In this section we learn how to create scripts that output to standard output using the standard library's `print` function, and how to run them with the Python interpreter.

5.1 The print function

The simplest way to get a Python program to do something is to print something.

The simplest thing to use is the `print` function in the standard library, section 2, *Built-in Functions*.

The library documentation shows this function's header as:

```
print(*objects, sep=' ', end='\n', file=sys.stdout,
      flush=False)
```

This needs some explanation:

- The name of the function is `print`.

- The arguments in parentheses are:

  `*objects` – zero or more values to print (All values are objects.)

  `sep=' '` – a keyworded parameter, defining the text that will be output between the objects, default is a space.

  `end='\n'` – the text that will be output after all of the objects, default is a newline.

  `file=sys.stdout` – the file to write to, default is standard output.

  `flush=False` – whether to flush the IO buffer on this file after output, default is no.

The keyworded parameters are optional.

5.2 Running a program that prints

An example program that prints.

It is saved in a file called `print1.py`.

```
# file: print1.py

print(42)      # print an int
print()        # print a blank line
print(4.1)     # print a float
print(42, 4.2) # print 2 things
```

This program is just using to `print` to output values to standard output.

Standard output is usually the console or terminal window.

Things to note about the program:

- Each line of the program is a call to the `print` function.
- Each line is a *statement*.
- In Python, statements do not need to be terminated with a semicolon (`;`).
- Comments start with a hash and extend to the end of the line.

Now we run it in our terminal or command prompt:

```
$ python3 print1.py
42

4.1
42 4.2
$
```

`print`'s default behaviour is to output 0 or more values separated by spaces, and ending with a newline.

A *newline* is the character (or characters) that cause the terminal to print subsequent text on the next line.

In this program, we use the end keyworded parameter to change the ending from a newline to other strings of our choosing.

```
# file: print2.py

print(42, end = ', ')
print(4.1, end = '; ')
print(42, 4.2)
```

```
$ python3 print2.py
42, 4.1; 42 4.2
$
```

In this program, we use the `sep` keyworded parameter to change the ending from a newline to add commas and the `end` keyworded parameter to end with a period and a newline.

(Note the sequence `\n` in a string makes a newline.)

```
# file: print3.py

print(1, 2, 3, 4, 5, sep = ', ', end = '.\n')
```

```
$ python3 print3.py
1, 2, 3, 4, 5.
$
```

In this program, demonstrate the values to print don't have to be simple literals. They can be any expression.

```
# file: print4.py
# Print the molecular weight of H2O.

print(2 * 1.00794 + 15.9994)
```

```
$ python3 print4.py
18.01528
$
```

Section summary

This section covered:

- the `print` function and some of how its optional keyworded arguments control the format of the output; and
- how to run a Python script with the interpreter.