LECTURE 12

COMMAND LINE ARGUMENTS

OPERATING SYSTEMS

MCS 260 Fall 2021

Emily Dumas

REMINDERS

- Worksheet 5 will be posted this afternoon
- Project 2 description coming soon

DOCSTRINGS

A Python function or file can begin with a string literal, a **docstring**, to document its purpose.

`help(function_name)` retrieves docstrings.

Any other explanatory text should be a comment.

A function with a docstring:

```python
def f(x):
    "Return the square of `x`"
    return x*x
```

Getting help for that function:

```
>>> help(f)
Help on function f in module __main__:

f(x)
    Return the square of `x`
>>>
```

EXPLANATORY TEXT

Is it the **first statement in a file**, or the **first statement in the body of a function**?

- Yes → Use a string literal, so it becomes a docstring
- No → Use a comment (#)

COMMAND LINE ARGUMENTS

Taking a break from Python, let's talk about the shell.

When you run a command in the shell, it may accept some strings as arguments, e.g.

```
PS C:\Users\ddumas> cd Desktop
PS C:\Users\ddumas\Desktop>
```

Here `cd` is the command name, and the string `Desktop` is the first (and only) **command line argument** (often called an **arg**).

Command line arguments are separated by spaces.

A Python program can access the command line arguments provided when it was run. For example, if a script is run with the command

```
python example.py now is the winter of our discontent
```

Then we can access each string after "python". This is useful so that a program can accept input from the command that runs it, rather than reading it from the keyboard.

To access command line args, we first import the `sys` module:

```
import sys
```

Now we have access to the list `sys.argv`. At index 0 it contains the name of our script (as given to the interpreter). At index 1 is the first argument after the script name, etc..

In the previous example, `sys.argv` would have value:

```
['example.py', 'now', 'is', 'the',
 'winter', 'of', 'our', 'discontent']
```

EXAMPLE

Let's write a program that repeats a message several times. It should take two command-line arguments:

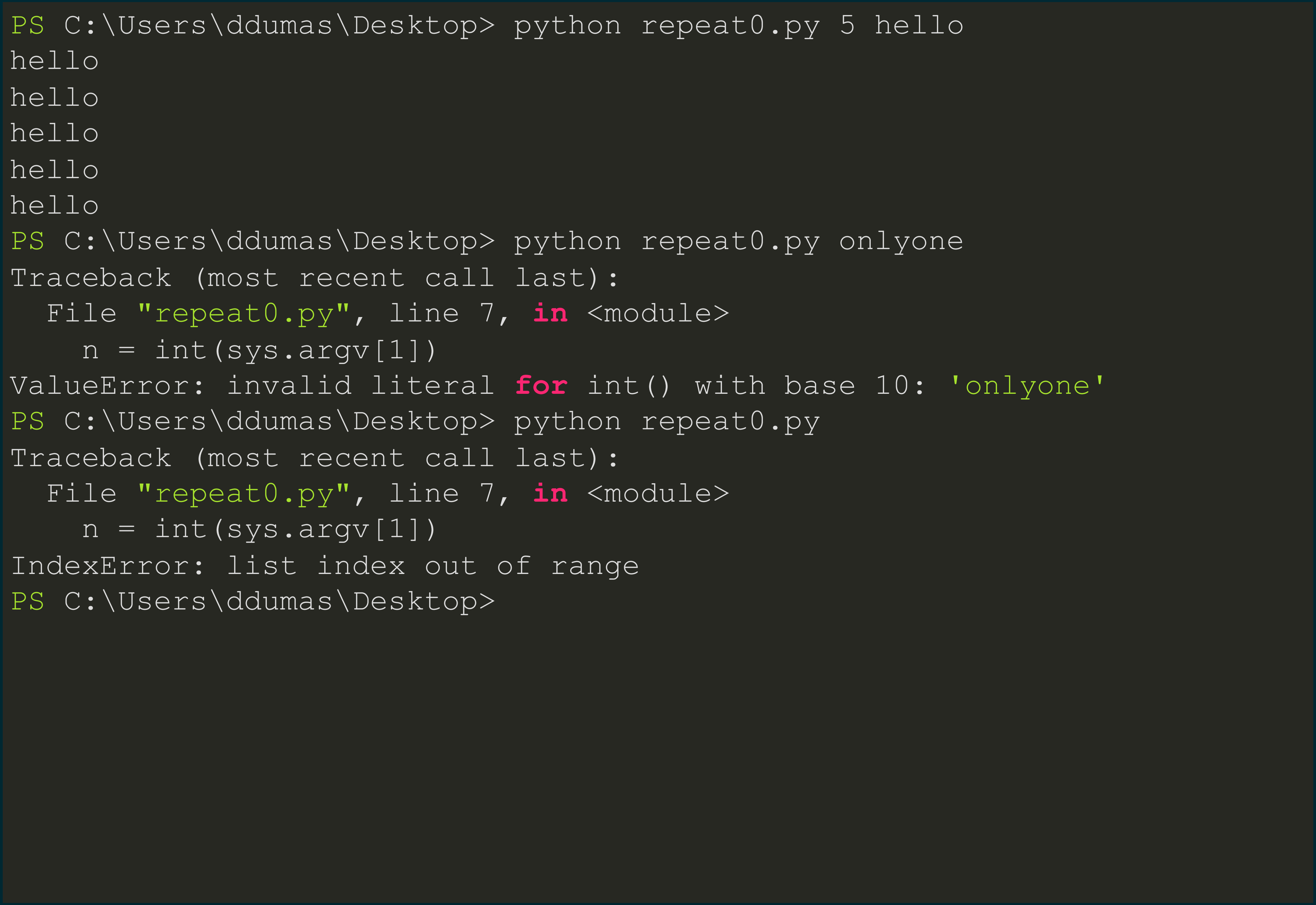
- Argument 1: Number of times
- Argument 2: Message

Basic version:

```python
"""Repeat a string a given number of times.
The first argument is the number of times.
The second gives the string to repeat.
"""

import sys
n = int(sys.argv[1])
s = sys.argv[2]

for i in range(n):
    print(s)
```



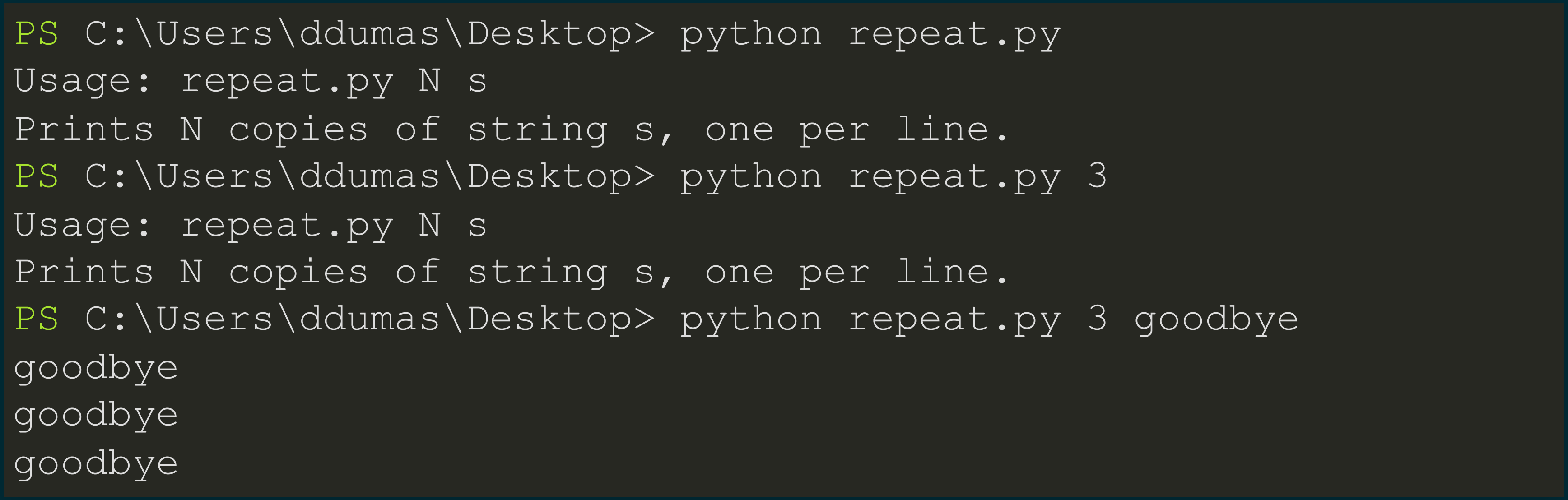
```
PS C:\Users\ddumas\Desktop> python repeat0.py 5 hello
hello
hello
hello
hello
hello
PS C:\Users\ddumas\Desktop> python repeat0.py onlyone
Traceback (most recent call last):
  File "repeat0.py", line 7, in <module>
    n = int(sys.argv[1])
ValueError: invalid literal for int() with base 10: 'onlyone'
PS C:\Users\ddumas\Desktop> python repeat0.py
Traceback (most recent call last):
  File "repeat0.py", line 7, in <module>
    n = int(sys.argv[1])
IndexError: list index out of range
PS C:\Users\ddumas\Desktop>
```

The better version repeat.py checks for too few arguments and handles it gracefully.

```python
"""Repeat a string a given number of times.
The first argument is the number of times.
The second gives the string to repeat.
"""

import sys

if len(sys.argv) < 3:
    print("Usage:",sys.argv[0],"N s")
    print("Prints N copies of string s, one per line.")
else:
    n = int(sys.argv[1])
    s = sys.argv[2]
    for i in range(n):
        print(s)
```



```
PS C:\Users\ddumas\Desktop> python repeat.py
Usage: repeat.py N s
Prints N copies of string s, one per line.
PS C:\Users\ddumas\Desktop> python repeat.py 3
Usage: repeat.py N s
Prints N copies of string s, one per line.
PS C:\Users\ddumas\Desktop> python repeat.py 3 goodbye
goodbye
goodbye
goodbye
```

The handoff of arguments from the shell to the Python script is one of the services of the **operating system** or OS.

Windows, Linux, Mac OS, Android, iOS are all operating systems.

An OS manages the lowest-level details of a computer's operation.

A key feature of operating systems is that they provide **abstraction**.

For example: A wireless mouse, a wired mouse, and a touchpad operate very differently. The OS handles these differences so that a program can ask for the current position of the pointer, without concern for the specific hardware.

GLOSSARY

- **CPU** - Central Processing Unit or processor. The main component of a computer that executes instructions in a computer program.
- **Hardware** - The physical parts (electronic devices) that make up a computer.
- **Software** - Collective term for computer programs.

GLOSSARY

- **RAM** - Random-Access Memory, or just memory. The place where currently-running programs and the data they use are stored. Variables are stored here. Contents of RAM are lost when the computer is powered off or restarted.
- **Persistent storage** - Hardware devices such as disks, USB flash drives, etc., that can store data that is not lost on restart or power-off.

SOME OS SERVICES

- **Device management**: communicate with attached devices (mouse, keyboard, disks, video controller, sound hardware) and provide a standardized interface for them.
- **Process management**: Control starting, stopping, running of programs as **processes**. Manage which processes have access to the CPU at a given time.

SOME OS SERVICES

- **Memory management**: Processes can only access parts of RAM that the OS allows them to. They can request access to more (or less) RAM.
- **File management**: Data stored on persistent storage devices is usually arranged into named **files**, which are in turn arranged into a hierarchy of **directories**. Storage devices know nothing about these concepts, and store only bytes. The OS provides the file/dir abstractions.

WHEN YOU CLICK "SAVE"

At a low level (hardware), assuming a wireless mouse:

- Your finger activates a switch in the mouse.
- A processor in the mouse is running a program that frequently checks the switch position. One such check notices it is closed, and calls a function to send notification of the change.
- The mouse begins sending bits of data using a 2.4Ghz radio signal.

- A bluetooth adapter in your computer that is constantly monitoring that radio frequency receives the data and asks for attention from the CPU.
- The CPU pauses the program it was running and switches to a driver in the OS that processes bluetooth data packets.
- The driver analyzes the raw data received over the radio link and adds a new "mouse event" to a list in RAM.

- Later, another part of the OS that notifies programs of user events gets access to the CPU. It begins searching for a process that should receive notification of the new mouse event.
- The editor window is identified as the recipient.
- Later, the editor asks the OS for new events, and gets the mouse event as an answer.

Skipping hundreds of steps until...

- The disk completes the request to write the bytes representing "...print('Hello world')\n" to the specified position.

WHEN YOU CLICK "SAVE"

At the level of OS-provided functions:

- A loop in the editor is constantly asking the OS if there are new events to handle. Eventually, it receives one—a mouse click.
- The editor determines the click is on "Save".
- A function within the editor to save the current file is called. It determines the filename, and asks the OS to open the file with permission to write.
- The OS gives the editor permission to write to that file.

- The editor takes the contents of the file the user is editing, encodes it into bytes, and asks the OS to write these bytes to the open file.
- The OS reports success, and the editor asks the OS to close the file.
- The editor updates its display to show that the file has no unsaved changes.

NEXT TIME

- Reading and writing files in Python
- String formatting

REFERENCES

- Command line arguments are not covered in the primary text.
- The documentation of sys.argv briefly summarizes today's material on command line arguments.
- Today's discussion of operating systems summarizes some of the material from Chapter 3 in Brookshear & Brylow.

REVISION HISTORY

- 2021-09-20 Initial publication