Managing the Filesystem with Python

Dr. Chris Brown

Calling Functions

Parameters and Arguments

Arguments

```
copyfile("foo", "bar")
```

```
def  copyfile( src, dst ):
    # code to copy
    # src to dst
```

Parameters

Positional Parameters

os.**putenv**(*key*, *value*)

Set the environment variable named *key* to the string *value*.

`os.putenv("EDITOR", "nano")`	✓ OK
`os.putenv("EDITOR")`	✗ Not enough arguments
`os.putenv("nano", "EDITOR")`	✗ Arguments in wrong order

Optional Parameters

`os.walk(top, topdown=True, onerror=None, followlinks=False)`

Generate the file names in a directory tree by walking the tree either top-down or bottom-up.

```
os.walk("/home/chris")
```

✓ OK — uses all defaults

```
os.walk("/etc", True, None, True)
```

✓ All arguments passed by position

```
os.walk("/etc", followlinks=True)
```

✓ One argument passed by keyword

Same

Keyword Parameters

`shutil.`**`copyfile`**(*src*, *dst*, *, *follow_symlinks=True*)

Copy the contents (no metadata) of the file named *src* to a file named *dst*

```
shutil.copyfile("foo", "bar")
```

✓ **OK, use default for** `follow_symlinks`

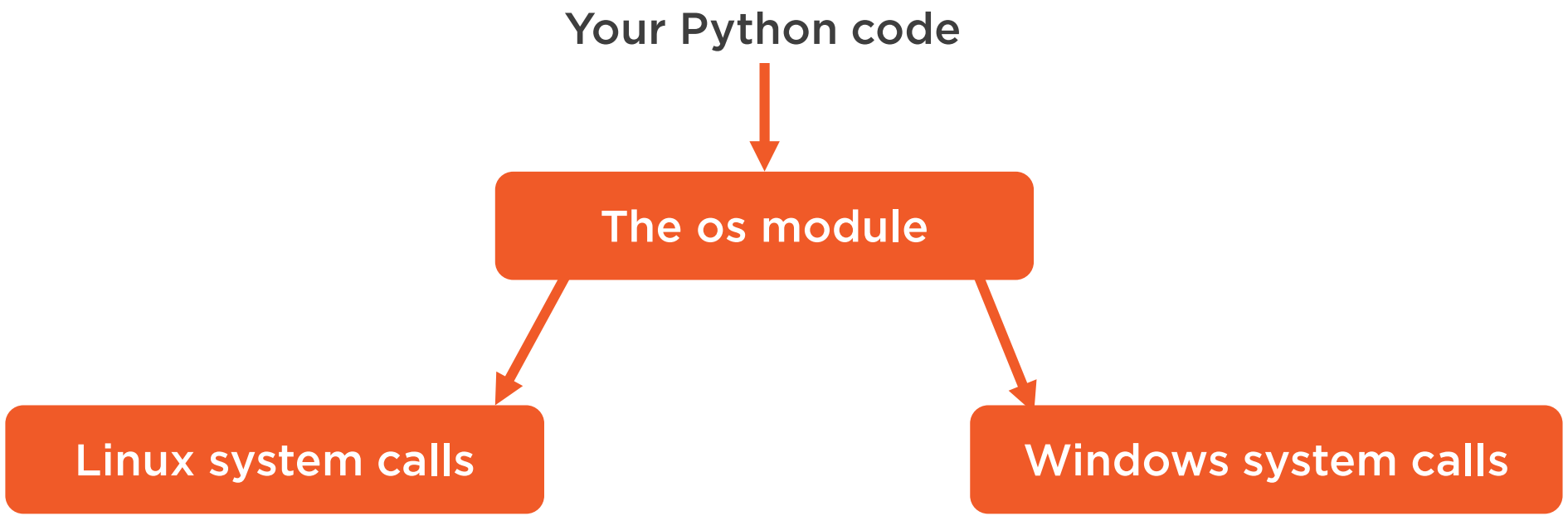
```
shutil.copyfile("foo", "bar", follow_symlinks=False)
```

✓ **OK**

```
shutil.copyfile("foo", "bar", False)
```

✗ **optional parameter is keyword-only**

The os Module

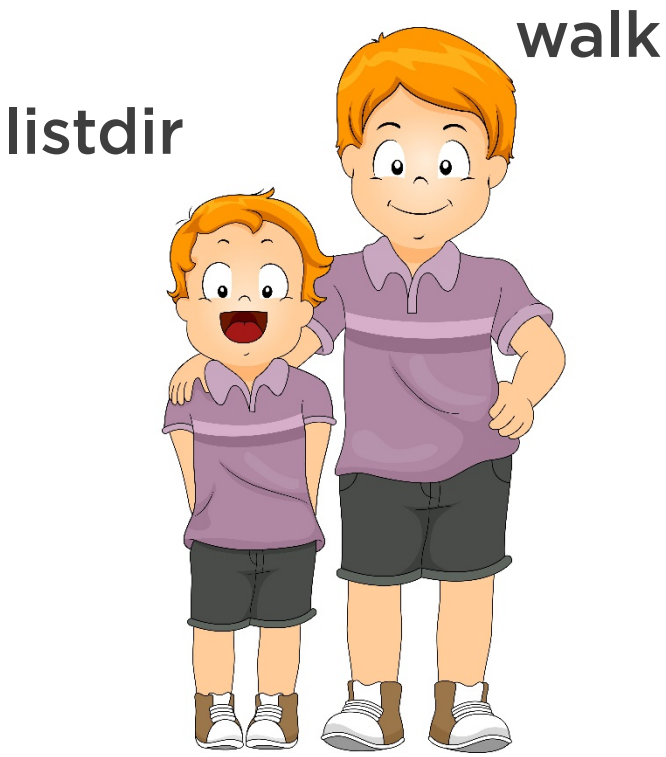


A Simple Directory Listing

```
!#/usr/bin/python3

import os

for file in os.listdir("."):
    info = os.stat(file)
    print("%-20s : size %d" % (file, info.st_size))
```



Walking the file system tree

`os.walk()` is `listdir`'s big brother

- Recursive traversal of the file system
- Generates a 3-tuple for each directory:
  - The pathname of the directory
  - A list of the directory names in it
  - A list of the file names in it

Walking the Filesystem Tree

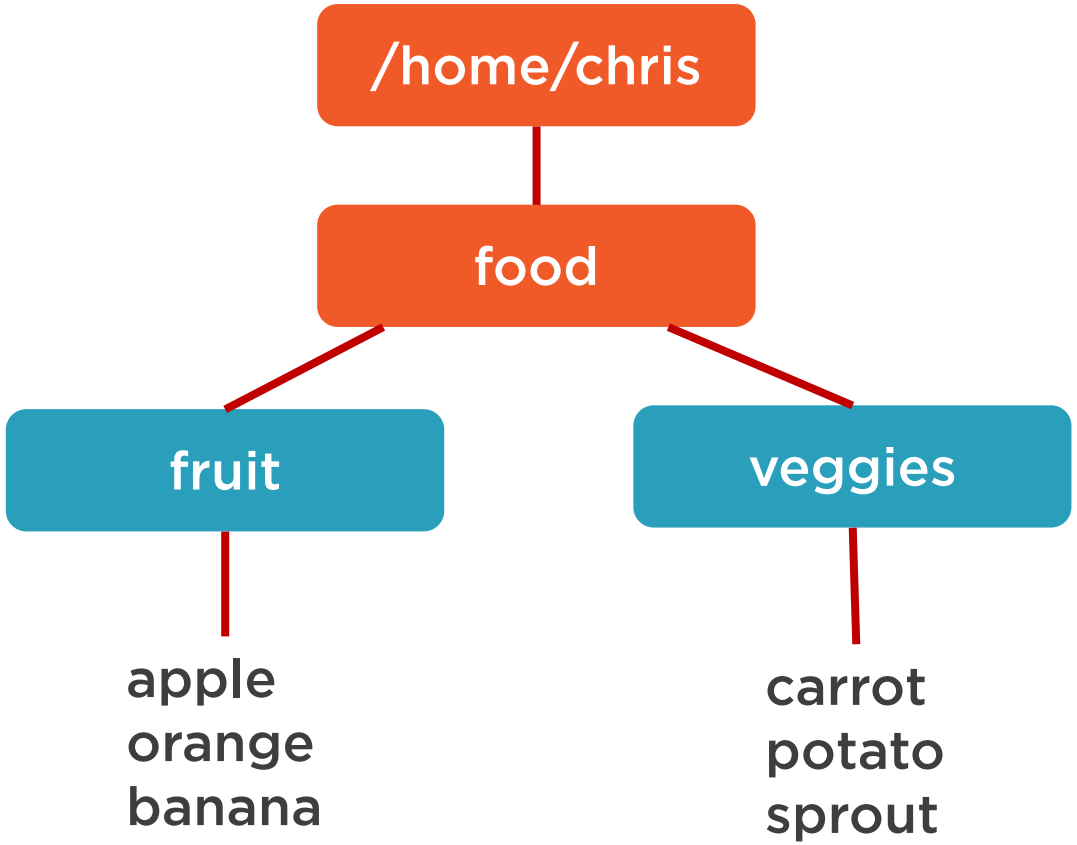
```
import os

for dirpath, dirnames, filenames in os.walk("/home/chris/food"):

    print("Files in %s are:" % dirpath)
    for file in filenames:
        print("\t" + file)

    print("Directories in %s are:" % dirpath)
    for dir in dirnames:
        print("\t" + dir)
```

The `food` Folder



The mission:

Find files that have no owner

How can this happen?

- Original owner's account deleted
- Import from "foreign" tar archive
- Deliberate change of owner: e.g.

```
# chown 9999 somefile
```

Step 1: Build a Set of UIDs

The direct approach

```
uidset = set()
for line in open("/etc/passwd"):
    split = line.split(":")
    uidset.add(int(split[2]))
```

Or using the pwd module ...

```
import pwd
uidset = set()
for user in pwd.getpwall():
    uidset.add(user.pw_uid)
```

Step 2: Walk the File System

```
import os

testdir = "/home/chris"
for folder, dirs, files in os.walk(testdir):
    for file in files:
        path = folder + "/" + file
        attributes = os.stat(path)
        if attributes.st_uid not in uidset:
            print(path + " has no owner")
```

Handling Broken Symlinks – Method 1

Make an explicit test and skip symlinks entirely:

```
for folder, dirs, files in os.walk(testdir):
    for file in files:
        path = folder + "/" + file

        if os.path.islink(path):
            print(path + " is a symlink ... skipping")
            continue

        attributes = os.stat(path)
        if attributes.st_uid not in uidset:
            print(path + " has no owner")
```

Handling Broken Symlinks – Method 2

Catch the exception thrown by `os.stat()` :

```
for folder, dirs, files in os.walk(testdir):
    for file in files:
        path = folder + "/" + file

        try:
            attributes = os.stat(path)
        except FileNotFoundError:
            print(path + " not found")
            continue

        if attributes.st_uid not in uidset:
            print(path + " has no owner")
```

os Revisited

```
remove(file)
```

```
chmod(file, mode)
```

```
mkdir(path [, mode])
```

```
chown(file, uid, gid)
```

```
rename(src, dst)
```

e.g. `0o644`

```
link(src, dst)
```

```
rmdir(path)
```

```
symlink(src, dst)
```

`shutil`: High Level File Operations

```
copy2(src, dst)
```

```
copy(src, dst)
```

```
rmtree(path)
```

```
copytree(src, dst, ignore=None)
```

```
move (src, dst)
```

```
which(cmd)
```

```
make_archive(basename, format)
```

```
unpack_archive(filename, extract_dir, format)
```

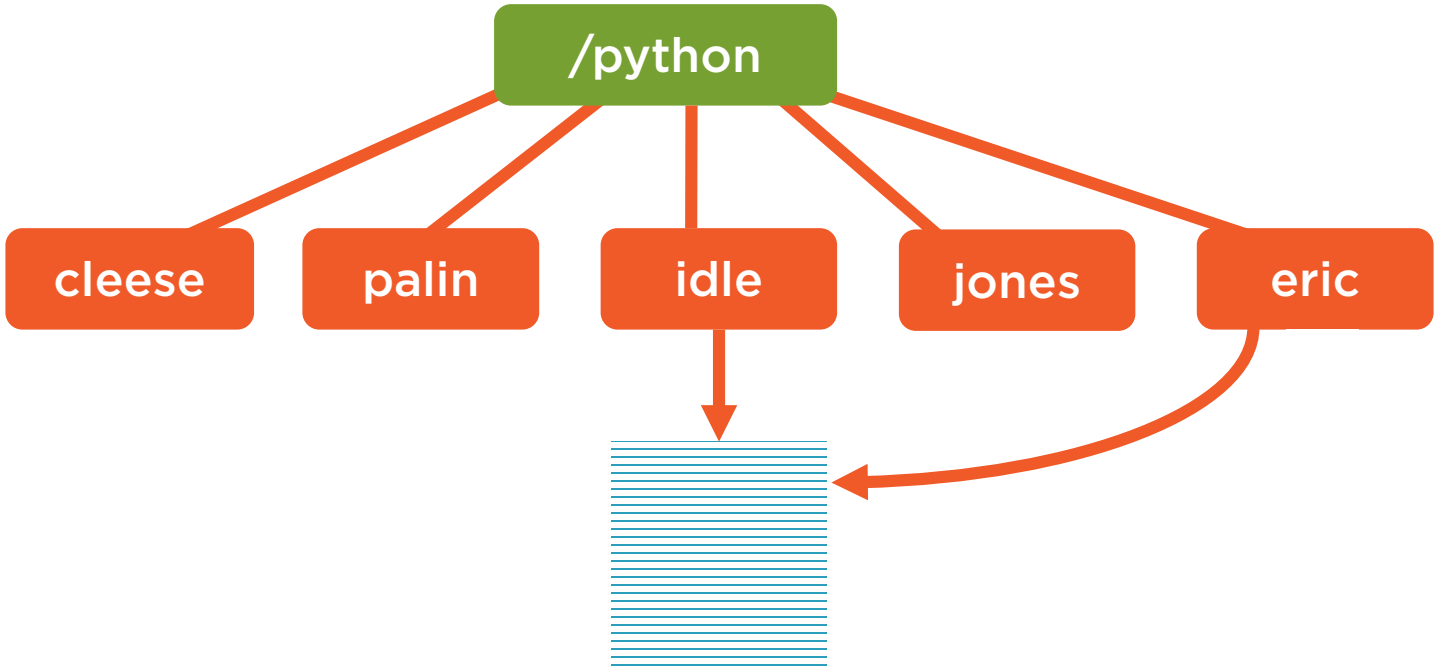
Dictionary

An unordered collection of key/value pairs

Also known as:

- associative array
- hash

De-duplication Strategy



Dictionary

Key	Value
6B52A0E771	/python/cleese
9DC7A402E5	/python/palin
6B52A0E771	/python/idle
1F9DE74463	/python/jones

Summary: Argument Passing

Positional

- `def func1(a, b)`

Optional / Keyword

- `def func2(a, b=0)`

Keyword only

- `def func3(a, *, b=0)`

Summary – Collections

Tuple

- An ordered set of immutable values
- `yellow = ( 200, 230, 0)`

List

- An ordered set of mutable values
- `users = [ "tom", "sue", "james", "mary" ]`

Dictionary

- An unordered set of key / value pairs
-
```
french = { "apple" : "pomme",
           "bread" : "pain",
           "wine"  : "vin"
         }
```

Summary: Modules

os

- Wrappers around Linux system calls
- Walking the file system

shutil

- High-level file operations

hashlib

- Secure hashes and message digests

Summary: Examples

- Simple directory listing
- Finding ownerless files
- File de-duplication

In the Next Lesson

Interacting with the Linux system

- Command line arguments
- Environment variables
- stdin, stdout, stderr
- Creating filters
- Handling signals