



COMP2720: Automating Tools for New Media

Functions: Why to use them,
what makes a good one, and
doing everything around
functions



Functions: What's the point?

- Why do we have functions?
- More specifically, why have more than one?
- And if you have more than one, which ones should you use?
- Once I have functions, what can I use them for?



Functions are for managing complexity

- Can we write all our programs as one large function?
YES, but it gets HARD!
- As programs grow in size, they grow in complexity.
 - How do you remember the details like inserting `<body>` and `` tags?
 - Put them inside of functions
 - How do you change the function over time and find the right place to make the changes you want?
 - If the function performs a specific role, then if you have to change that role, you change that function.
 - How do you test and debug your program?
 - You can put print statements in the whole thing, or, you can test individual functions first.



Advantages to using functions

- Hides details so that you can ignore them.
- Makes testing easier.
- Helps you in writing new programs because you can *reuse* trusted, useful functions.

Example: Generating a home page

```
def makeHomePage(name, interest):
    file=open("homepage.html","wt")
    file.write("""<!DOCTYPE HTML PUBLIC "-
//W3C//DTD HTML 4.01 Transition//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>"""+name+""""s Home Page</title>
</head>
<body>
<h1>Welcome to """+name+""""s Home Page</h1>
<p>Hi! I am """+name+"""". This is my home page!
I am interested in """+interest+""""</p>
</body>
</html>""")
    file.close()
```

Which one of these programs is simpler?

```
def makeHomePage(name, interest):
    file=open("homepage.html","wt")
    file.write(doctype())
    file.write(title(name+"s Home Page"))
    file.write(body("""
<h1>Welcome to """+name+""""s Home Page</h1>
<p>Hi! I am """+name+"""". This is my home page!
I am interested in """+interest+""""</p>"""))
    file.close()

def doctype():
    return '<!DOCTYPE HTML PUBLIC "-
//W3C//DTD HTML 4.01 Transition//EN"
"http://www.w3.org/TR/html4/loose.dtd">'

def title(titlestring):
    return
    "<html><head><title>"+titlestring+"</title></head>"

def body(bodystring):
    return "<body>"+bodystring+"</body></html>"
```

Focusing on the part that we would most likely change

```
def makeHomePage(name, interest):
    file=open("homepage.html","wt")
    file.write("""<!DOCTYPE HTML PUBLIC "-
//W3C//DTD HTML 4.01 Transition//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>"""+name+""""s Home Page</title>
</head>
<body>
<h1>Welcome to """+name+""""s Home Page</h1>
<p>Hi! I am """+name+"""". This is my home page!
I am interested in """+interest+""""</p>
</body>
</html>""")
    file.close()
```

```
def makeHomePage(name, interest):
    file=open("homepage.html","wt")
    file.write(doctype())
    file.write(title(name+"s Home Page"))
    file.write(body("""
<h1>Welcome to """+name+""""s Home Page</h1>
<p>Hi! I am """+name+"""". This is my home page!
I am interested in """+interest+""""</p>"""))
    file.close()
```

Now which one is simpler?

Simpler to change? Simpler to add to?



Make *testing* simpler

- We can now check the individual pieces, rather than only the whole thing.
- If the individual pieces work, it's more likely that the whole thing works.
 - You still have to make sure that the pieces fit together well (called *integration testing*), but you already know what the pieces do and if the pieces work.

Example: Testing the pieces

```
>>> print doctype()
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transition//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

```
>>> print title("My title string")
```

```
<html><head><title>My title string</title></head>
```

```
>>> print body("<h1>My heading</h1><p>My paragraph</p>")
```

```
<body><h1>My heading</h1><p>My  
paragraph</p></body></html>
```



Adding functions makes it simpler if the functions are chosen well

- What if we had sub-functions that did smaller pieces of the overall task?
 - We call that changing the *granularity*.
- Is that better or worse?
 - It's better if it makes the overall program easier to understand and to change.
 - It's worse if it simply swaps one kind of complexity for another
 - Remembering all those functions and having to call lots of functions rather than the one that does what we need.



Your goal with testing functions: Trust

- Do you know what a function is *supposed* to do?
 - Do you really understand it?
- Does it do what you expect it to do?
- For whatever input you give it?
 - Key: Can you now forget about how it works and just assume that it does work?

Using *subfunctions* to ease testing and complexity

- Have a look at this program

```
import os
```

```
def makeSamplePage(directory):
    samplesfile=open(directory+"//samples.html","wt")
    samplesfile.write(doctype())
    samplesfile.write(title("Samples from "+directory))
    # Now, let's make up the string that will be the body.
    samples="<h1>Samples from "+directory+" </h1>\n"
    for file in os.listdir(directory):
        if file.endswith(".jpg"):
            samples=samples+"<p>Filename: "+file+"<br>"
            samples=samples+'</p>\n'
    samplesfile.write(body(samples))
    samplesfile.close()
```



What's the hard part? That loop body!

- Useful *heuristic* (rule of thumb):
If it's hard, break it out into a *subfunction* so that you can debug and fix that part on its own.

Breaking out the loop body

```
def makeSamplePage(directory):
    samplesfile=open(directory+"//samples.html", "wt")
    samplesfile.write(doctype())
    samplesfile.write(title("Samples from "+directory))
    # Now, let's make up the string that will be the body.
    samples=""<h1>Samples from "+directory+" </h1>\n"
    for file in os.listdir(directory):
        if file.endswith(".jpg"):
            samples = samples + fileEntry(file)
    samplesfile.write(body(samples))
    samplesfile.close()

def fileEntry(file):
    samples=""<p>Filename: "+file+"<br>"
    samples=samples+'</p>\n'
    return samples
```

Use more lines, if you want

```
def makeSamplePage(directory):  
    samplesfile=open(directory+"//samples.ht  
ml", "wt")  
    samplesfile.write(doctype())  
    samplesfile.write(title("Samples from  
"+directory))  
    # Now, let's make up the string that will be  
    the body.  
    samples="<h1>Samples from "+directory+"  
</h1>\n"  
    for file in os.listdir(directory):  
        if file.endswith(".jpg"):  
            samples = samples + fileEntry(file)  
    samplesfile.write(body(samples))  
    samplesfile.close()
```

```
def fileEntry(file):  
    samples="<p>Filename: "  
    samples=samples+file  
    samples=samples+"<br>"  
    samples=samples+'</p>\n'  
    return samples
```

If it makes the code more readable to you, do it that way!

Testing it by itself

```
>>> print fileEntry("barbara.jpg")
```

```
<p>Filename: barbara.jpg<br></p>
```

```
>>> print fileEntry("sunset.jpg")
```

```
<p>Filename: sunset.jpg<br></p>
```



Changing the program considerably

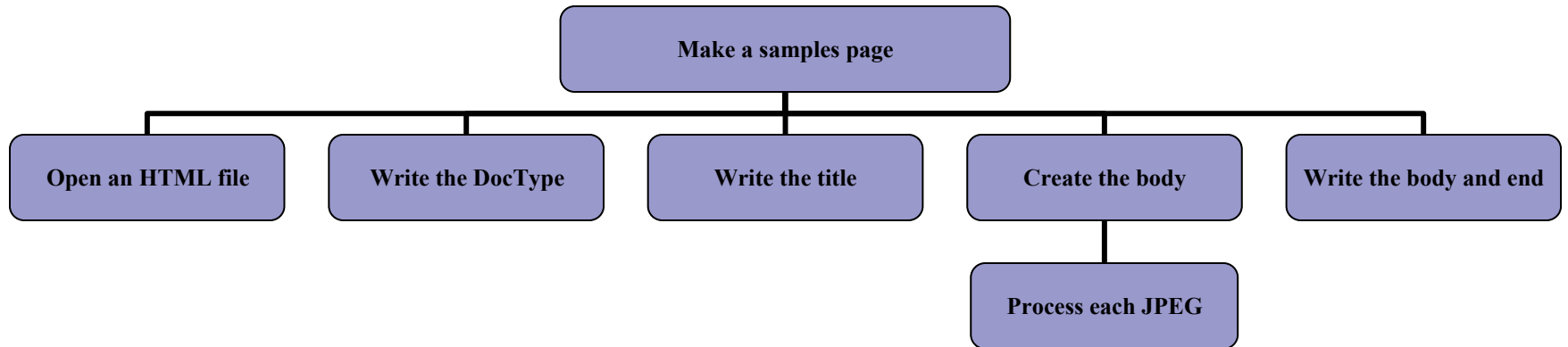
- What if we want to process pictures and sounds separately?
- How do we think about that?
- We use a process called *procedural abstraction*.



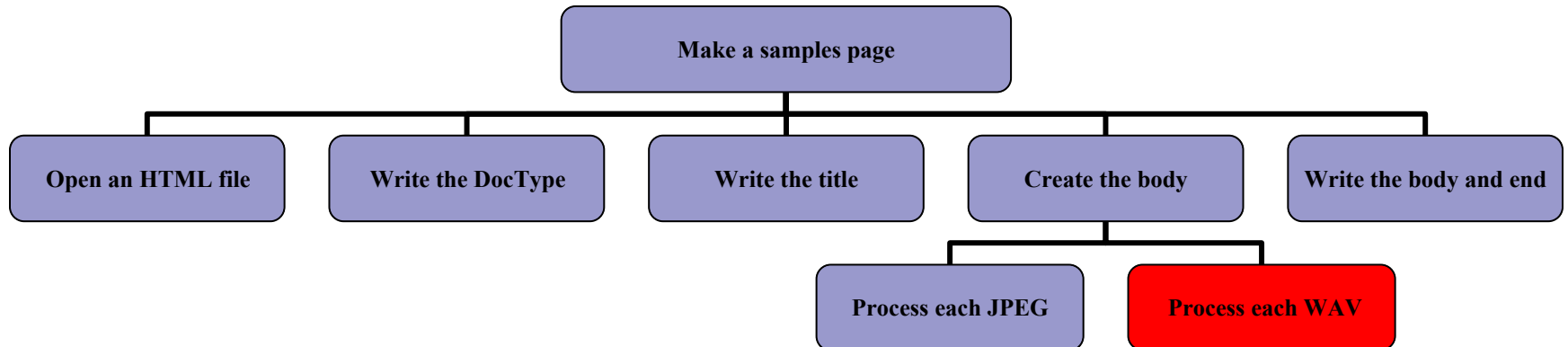
Procedural abstraction

- State the problem.
- Break the problem into sub-problems.
- Keep breaking the sub-problems into smaller problems until you know how to write that chunk.
- Goal: Main function is basically telling all the sub-functions what to do.
 - Each sub-function does one logical task.

What are the problems and sub-problems we're solving now?



What we want to change: Processing WAV files, too




Version 1: Not too different

```
def makeSamplePage(directory):
    samplesfile=open(directory+"//samples.html", "wt")
    samplesfile.write(doctype())
    samplesfile.write(title("Samples from "+directory))
    # Now, let's make up the string that will be the body.
    samples=""<h1>Samples from "+directory+" </h1>\n"
    for file in os.listdir(directory):
        if file.endswith(".jpg"):
            samples = samples + fileJPEGEntry(file)
        if file.endswith(".wav"):
            samples=samples+fileWAVEntry(file)
    samplesfile.write(body(samples))
    samplesfile.close()
```

```
def fileJPEGEntry(file):
    samples=""<p>Filename: "+file
    samples=samples+"<br>"
    samples=samples+'</p>\n'
    return samples
```

```
def fileWAVEntry(file):
    samples=""<p>Filename: "
    samples=samples+file
    samples=samples+'</p>\n'
    return samples
```



What if we wanted to compute sizes of pictures and sounds, too?

- We'll have to pass the directory as input to the lower functions.
 - Because now we have to find the actual file.
- Code gets a little more complicated.

Main function

```
def makeSamplePage(directory):
    samplesfile=open(directory+"//samples.html","wt")
    samplesfile.write(doctype())
    samplesfile.write(title("Samples from "+directory))
    # Now, let's make up the string that will be the body.
    samples("<h1>Samples from "+directory+" </h1>\n"
    for file in os.listdir(directory):
        if file.endswith(".jpg"):
            samples = samples + fileJPEGEntry(directory, file)
        if file.endswith(".wav"):
            samples=samples+fileWAVEntry(directory, file)
    samplesfile.write(body(samples))
    samplesfile.close()
```

WAV and JPEG file entry functions

```
def fileJPEGEntry(directory, file):
    samples="<p>Filename: "+file
    samples=samples+"<br>"
    samples=samples+''
    pic = makePicture(directory+"//"+file)
    samples=samples+" Height:
    "+str(getHeight(pic))
    samples=samples+" Width:
    "+str(getWidth(pic))
    samples=samples+'</p>\n'
    return samples
```

```
def fileWAVEntry(directory, file):
    samples="<p>Filename: "
    samples=samples+file+", "
    sound=makeSound(directory+"//"+file)
    length = getLength(sound) /
    getSamplingRate(sound)
    samples=samples+"Length (seconds):
    "+str(length)
    samples=samples+'</p>\n'
    return samples
```

Running the new program


Samples from C:\Documents and Settings\Mark Guzdial\My Documents\mediasources\pics

Filename: [aah.wav](#) Length (seconds): 1.9504761904761905



Filename:  Height: 535 Width: 276

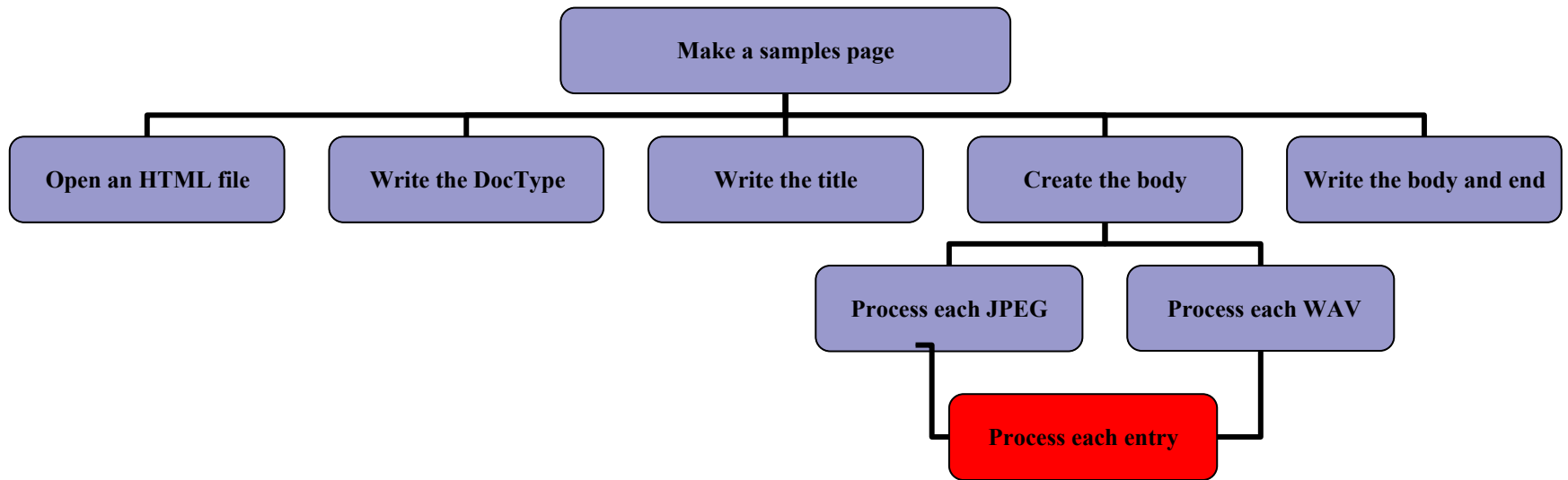


Filename:  Height: 409 Width: 292

Not really *modular*

- In a *modular* program (a program that has “good modularity”) each function does one and only one task well.
 - Look at all the duplicated code in the two file entry functions.
 - Could we pull that out into yet another function?

Creating a sub-sub-function



Pulling out the sub-sub-function

```
def fileJPEGEntry(directory, file):
    samples='<br> '
    pic = makePicture(directory+"//"+file)
    samples=samples+" Height: "+str(getHeight(pic))
    samples=samples+" Width: "+str(getWidth(pic))
    return fileEntry(samples, file)
```

```
def fileWAVEntry(directory, file):
    sound=makeSound(directory+"//"+file)
    length = getLength(sound)/getSamplingRate(sound)
    samples=", Length (seconds): "+str(length)
    return fileEntry(samples, file)
```

```
def fileEntry(filestring, file):
    samples="<p>Filename: "+file
    samples=samples+filestring
    samples=samples+"</p>\n"
    return samples
```

We can now test functions separately

```
>>> print fileEntry("Here is a file","picture.jpg")
```

```
<p>Filename: picture.jpg Here is a file</p>
```

```
>>> print fileWAVEEntry(r"C:\Documents and Settings\Mark Guzdial\My Documents\mediasources","aah.wav")
```

```
<p>Filename: aah.wav, Length (seconds): 1.9504761904761905</p>
```

```
>>> print fileJPEGEntry(r"C:\Documents and Settings\Mark Guzdial\My Documents\mediasources","barbara.jpg")
```

```
<p>Filename: barbara.jpg<br>   
Height: 294 Width: 222</p>
```



Want it in a list?

- What if you wanted each file to be an item in an unordered list?
- Four changes, basically.
 - Add the `` and `` to the main function.
 - Add the `` and `` to each file entry.

Adding the list items

```
def makeSamplePage(directory):
```

```
    samplesfile=open(directory+"//samples.html", "wt")
    samplesfile.write(doctype())
    samplesfile.write(title("Samples from "+directory))
    # Now, let's make up the string that will be the body.
    samples="<h1>Samples from "+directory+"</h1><ul>\n"
    for file in os.listdir(directory):
        if file.endswith(".jpg"):
            samples = samples +
                fileJPEGEntry(directory, file)
        if file.endswith(".wav"):
            samples=samples+fileWAVEntry(directory,
                file)
    samplesfile.write(body(samples+"</ul>"))
    samplesfile.close()
```

```
def fileEntry(filestring,file):
    samples="<p><li>Filename: "
    samples=samples+file
    samples=samples+filestring
    samples=samples+"</li></p>\n"
    return samples
```

Testing the new version

Samples from C:\Documents and Settings\Mark Guzdial\My Documents\mediasources\pics

- Filename: [aah.wav](#) Length (seconds): 1.9504761904761905



- Filename: [aah.wav](#) Height: 535 Width: 276



- Filename: [aah.wav](#) Height: 409 Width: 292



Notice that both sounds and pictures are updated.



Reusability: The reason why professionals value modularity

- When a function does one and only one thing, it can easily be reused in new situations.
 - Consider the functions you used in labs and assignment 1.
 - Remember those functions that also showed a picture at the end.
 - They literally couldn't be used for other programs, because you'd have many pictures popping up.
- Professionals will create a library of their own reusable functions that they'll use in their work.
 - That's why we have modules and the import statement: To make that kind of library easier to use.



Summary: Why we use functions

- Hides details so that you can ignore them.
- Makes testing easier.
- Helps you in writing new programs because you can reuse trusted, useful functions.