Python Overview 4:

Additional topics

Presentation Overview

- Running shell commands
- Formatting strings
- User input

Running shell commands

- You can run shell commands within Python and capture the output
- Very handy for many DevOps tasks

Running shell commands

- Use the subprocess module

  `subprocess.run()`

- There are many usage variations, e.g.

```
$ mkdir demo; cd demo; touch testfile1 testfile2 testfile3
$ python3
>>> import subprocess
>>> result = subprocess.run("pwd")
/home/jbloggs/demo
>>> print(result.returncode)
0                                          (0 normally indicates success)
>>> result = subprocess.run(["ls", "-l"])
-rw-rw-r--  1 jbloggs  jbloggs  0 Jan 16 10:37 testfile1
-rw-rw-r--  1 jbloggs  jbloggs  0 Jan 16 10:37 testfile2
-rw-rw-r--  1 jbloggs  jbloggs  0 Jan 16 10:37 testfile3
>>>
```

Running shell commands – security note

- Beware that running shell commands from within a program can introduce security threats
  - If exposed directly or indirectly to untrusted users (e.g. via a web app)
- Risk of *command injection attack*
- Imagine we have a variable *filename* populated from a user web form and the code

```
>>> result = subprocess.run("cat " + filename)
```

- Consider what happens if filename comes from malicious input such as

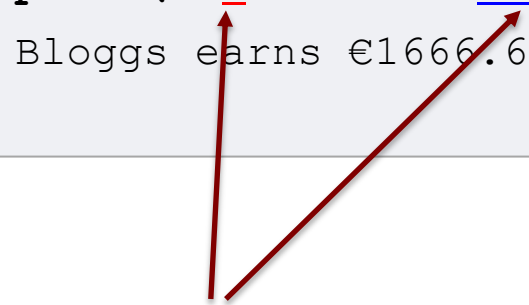
testfile; /bin/rm –rf /

Warning: don't test with this example!!

String Formatting: old "C-style"

- Similar to *printf* in the C programming language
- *<formatted string> % <elements to insert>*
- Can usually just use %s for everything – this will convert the object to its String representation.

```
>>> employee = 'Joe Bloggs'
>>> salary = 20000
>>> print("%s earns €%.2f per month" % (employee, salary/12))
Joe Bloggs earns €1666.67 per month
>>>
```



String Formatting – string.format()

- New alternative way to format strings introduced in version 3.6
- Similar to C-style but tidier syntax
- More flexible with named parameters (can change order for example)

```
>>> employee = 'Joe Bloggs'
>>> salary = 20000
>>> print("{0:s} earns €{1:.2f} per month".format(employee,
                                                  salary/12))
Joe Bloggs earns €1666.67 per month
>>> print("{name:s} earns €{pay:.2f} per month".format(name=employee,
                                                       pay=salary/12))
Joe Bloggs earns €1666.67 per month
>>>
```

String Formatting – new *"f-strings"*

- Introduced in Python 3.6
- Less verbose / more compact code

```
>>> employee = 'Joe Bloggs'
>>> salary = 20000
>>> print(f"{employee:s} earns {salary/12:.2f} per month")
Joe Bloggs earns €1666.67 per month
```

Input

- The **input**(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods **int**(string), **float**(string), etc.

Input example

input.py

```
print ("What's your name?")
name = input("> ")
print ("What year were you born?")
birthyear = int(input("> "))
print ("Hi %s! You are %d years old!" % (name,
2019 - birthyear))
```

```
$ python3 input.py
What's your name?
> Michael
What year were you born?
> 1985
Hi Michael! You are 34 years old!
$
```