

# Developer Operations

---

## Python Overview 1: Getting Started

Credits: parts extracted from presentations by Moshe Goldstein and Michael DiRamio

# Presentation Overview

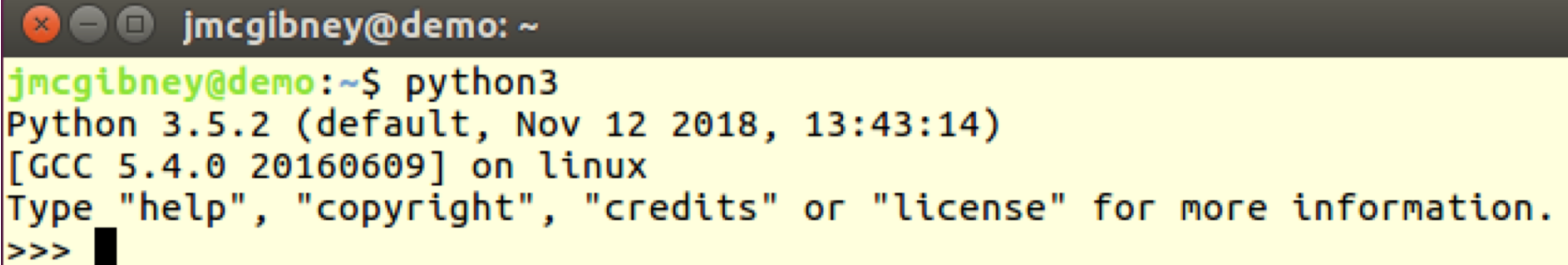
---

- Running Python
- Variables
- Basic data types
- Control flow

# Installation / set-up

- Python 3 comes with most Linux distributions and can be easily installed on Mac OS X
  - Just open a terminal window and type “**python3**”
  - You’ll get a prompt like this:

>>>

A terminal window with a dark gray title bar containing window control icons and the text 'jmcgibney@demo: ~'. The terminal has a light yellow background. It shows the command 'python3' being executed, followed by the Python version and compiler information, and a prompt '>>>' with a cursor.

```
jmcgibney@demo: ~  
jmcgibney@demo:~$ python3  
Python 3.5.2 (default, Nov 12 2018, 13:43:14)  
[GCC 5.4.0 20160609] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

# Installation / set-up

---

- Can get Windows version, but better to run Python on Linux (in a virtual machine is fine). Mac is also fine.
- Choose Python 3 rather than Python 2
  - Latest stable version is 3.7.2 (at time of writing) but any recent version should be ok.
  - Python 3 comes preinstalled on many Linux distributions. The command is *python3*  
\$ **python3**
  - For Amazon Linux, Python 3 first needs to be installed  
\$ **sudo yum install python37**  
\$ **python3**

# Documentation

---

- Documentation home:  
<https://docs.python.org/3/>
  - Tutorial:  
<https://docs.python.org/3/tutorial/>
  - Reference manual:  
<https://docs.python.org/3/reference/>
- + lots more

# The Python Interpreter

- Python is an interpreted language
- The interpreter provides an interactive environment to play with the language
  - Really useful for trying out syntax
- Results of expressions are printed on the screen

```
>>> 3 + 7
10
>>> 3 < 15
True
>>> 'print me'
'print me'
>>> print ('print me')
print me
>>>
```

# Interactive Python – Hello World

---

- At the Python >>> prompt, type 'Hello world!'

*Interactive  
Python  
prompt*



```
>>> 'Hello world!'  
Hello world!
```

- Or alternatively:

```
>>> print ('Hello world!')  
Hello world!
```

# Or put it in a script/program

- ... to make your code reusable
- Use an editor to create a file called helloworld.py and type in a line of code containing the call to print()

*Linux prompt* → `$ nano helloworld.py`  
*(enter code: print ('Hello world!'))*  
`$ cat helloworld.py`  
`print ('Hello world!')`  
`$ python3 helloworld.py`  
`Hello world!`



# The print Statement

---

- Elements separated by commas print with a space between them

```
>>> print ('Hello')
Hello
>>> print ('Hello', 'there')
Hello there
>>>
```

# Comments

---

**# starts a comment**

```
print ('This will print') # comment here  
#print ('This will not')
```

# Variables

---

- Variables are not declared, just assigned
- The variable is created the first time you assign it a value
- Variables are references to objects
- Type information is with the object, not the reference
- Everything in Python is an object

# All variables are objects

---

- Everything is an object
- Data type is a property of the *object* and not of the variable

```
>>> x = 7
>>> print(x)
7
>>> x = 'Hello'
>>> print(x)
'Hello'
>>>
```

# Numbers

- Python has integers, long integers and floating point numbers (plus others types like complex numbers)

```
>>> 132224
132224
>>> 132323 ** 4
306578259430545516241
>>> 1.23232
1.23232
>>> print (1.23232)
1.23232
>>> 1.3E7
13000000.0
>>>
```

# String Literals

- Strings are *immutable*
  - *i.e. they can't be changed*
  - *we just create a new string when we carry out an operation*
- **+** is overloaded to do concatenation

```
>>> x = 'hello'
>>> x = x + ' there'
>>> print (x)
'hello there'
>>>
```

Here a **new** string is created  
and assigned to variable *x*

- Short video on string immutability:  
<https://www.youtube.com/watch?v=LTw5-5tx5wg>

# String Literals: many kinds

---

- Can use single or double quotes, and three double quotes for a multi-line string

```
>>> print('I am a string')
'I am a string'
>>> print ("So am I!")
'So am I!'
>>> s = """And me too,
though I am much longer
than the others"""
>>> print (s)
And me too,
though I am much longer
than the others
```

# Booleans

---

- The following are false:
  - 0
  - *None*
  - *False*
  - An empty string, list, tuple, or dictionary
- All other values are considered true



# Boolean Expressions

---

- Boolean expressions can be evaluated directly by the interpreter
- Note that when *None* is returned the interpreter does not print anything

```
>>> True and False
False
>>> False or True
True
>>> None and 2
>>> None or 2
2
>>>
```

# No braces – i.e. no { }

---

- Python uses **indentation** instead of braces { } to determine the scope of expressions
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope)
- This forces the programmer to use proper indentation since the indenting is part of the program!

# Control flow: if

---

```
x = 20
y = 30
if x < y :
    print ('x is less than y')
elif x > y :
    print ('x is greater than y')
else :
    print ('x is equal to y')
```

# while loops

---

```
x = 1
while x < 5 :
    print (x)
    x = x + 1
```

whileloop.py

```
$ python3 whileloop.py
1
2
3
4
$
```

Running in a shell

# for loops

- Iterates through a list of values

forloop1.py

```
for x in [1,7,13,2]:  
    print (x)
```

```
$ python forloop1.py
```

```
1  
7  
13  
2  
$
```

forloop2.py

```
for x in range(5) :  
    print (x)
```

```
$ python forloop2.py
```

```
0  
1  
2  
3  
4  
$
```

*range(N) generates a list of numbers [0,1, ..., n-1]*