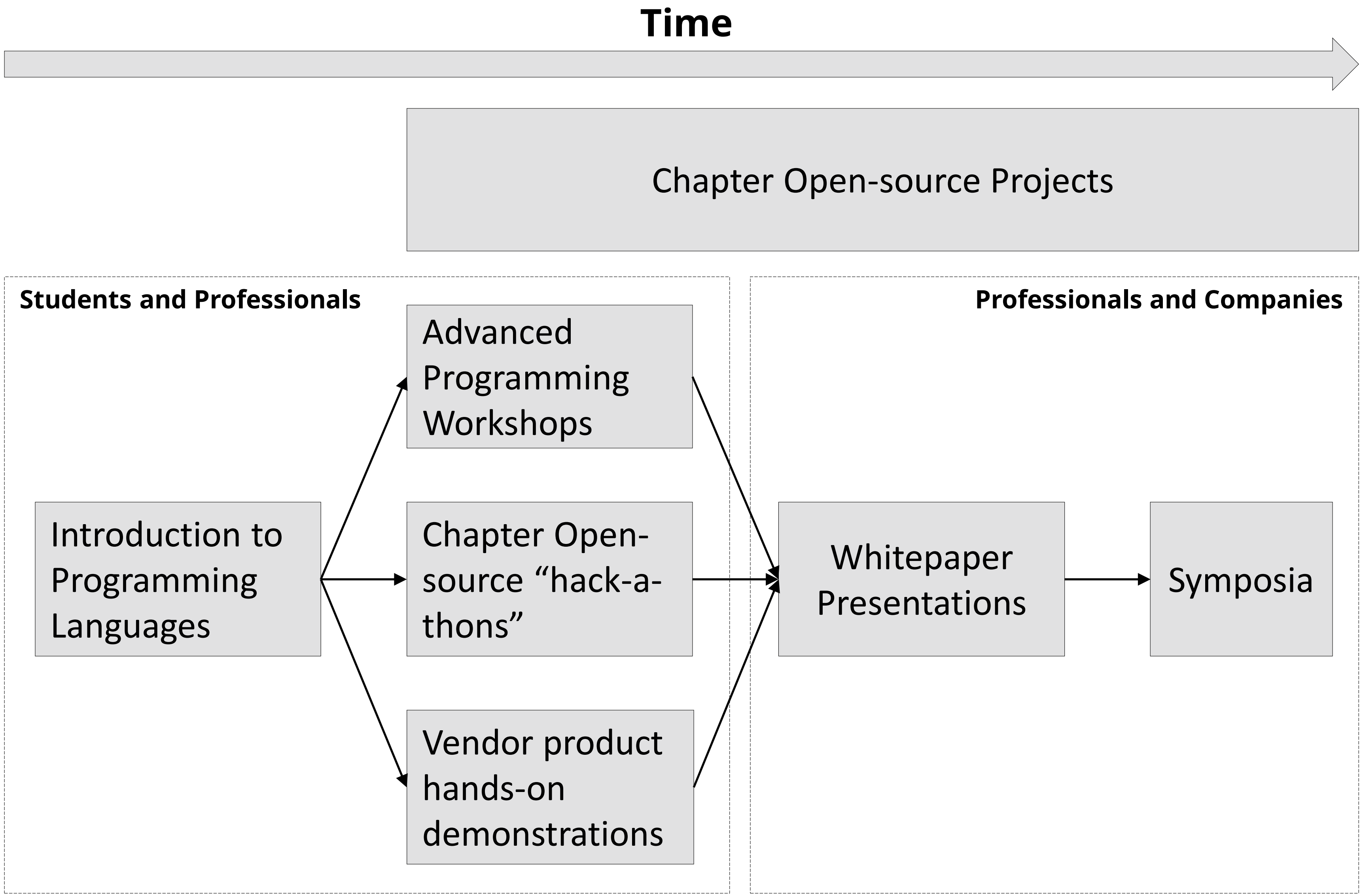
Introduction to Python for Engineers and Manufacturers

Adam J. Cook, Chair of SME Chapter 112

FOR PUBLIC RELEASE

Chapter “Digital” Initiative


Time
Chapter Open-source Projects
Students and Professionals
Introduction to Programming Languages
Advanced Programming Workshops
Chapter Open-source “hack-a-thons”
Vendor product hands-on demonstrations
Professionals and Companies
Whitepaper Presentations
Symposia

FOR PUBLIC RELEASE

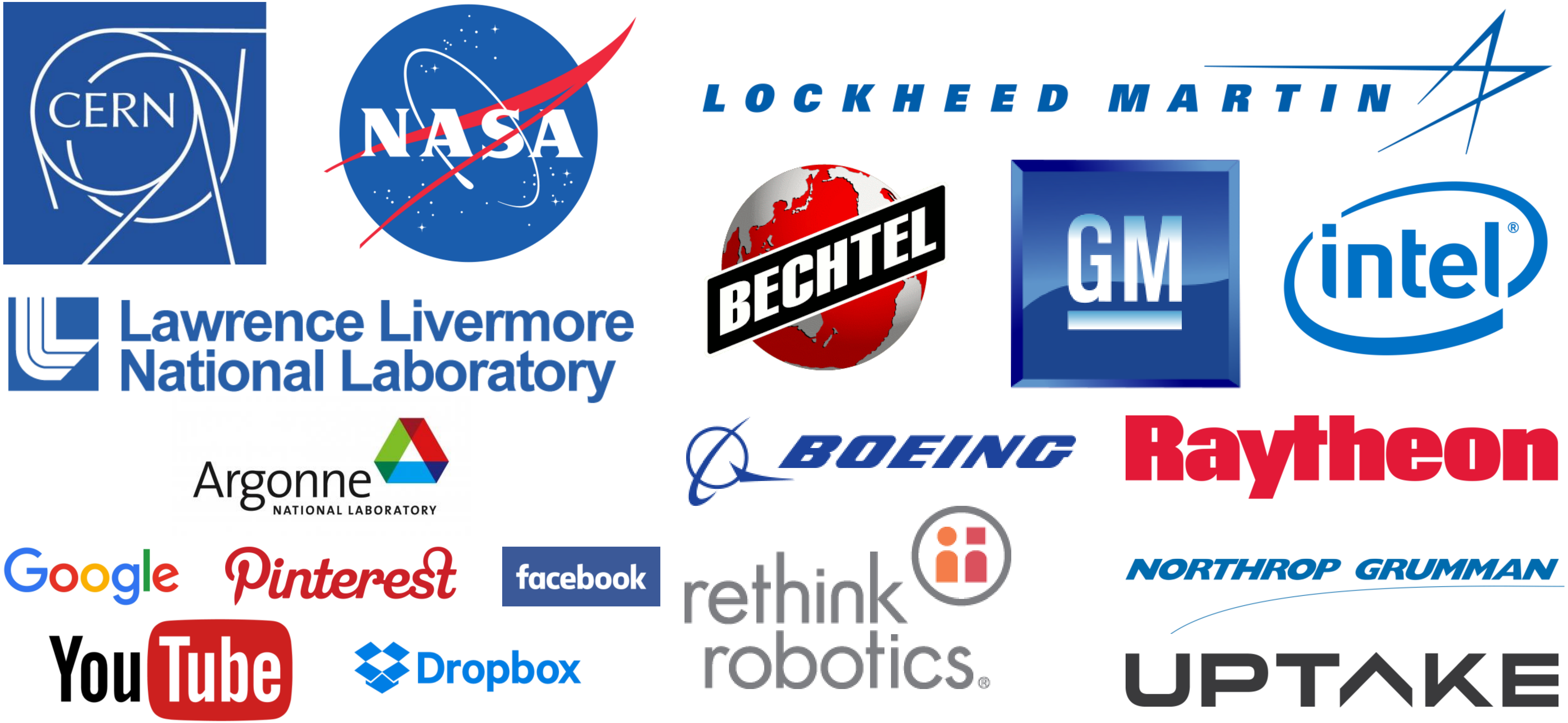
What is Python?

- High-level programming language.
- Free and open-source.
- Interpreted.
- Cross-platform.
- Extensive standard library.
- Automatic memory management.
- Designed to be highly readable and explicit.
- Proven to be quite versatile (and popular).
- Reasonably fast for many applications.
- **Why not just use MATLAB?**

What does “open-source” mean?

- The Open Source Definition - https://opensource.org/osd-annotated
- Free redistribution, royalty-free.
- Source code availability.
- Popular open-source licenses – MIT, BSD, Apache 2.0, GPL, LGPL.
- The use of open-source components are ubiquitous in industry.
- **Always** check with your legal representation if a non-standard license is encountered or if you are unsure of your legal rights and responsibilities with standard OSS licenses.

Who uses Python?



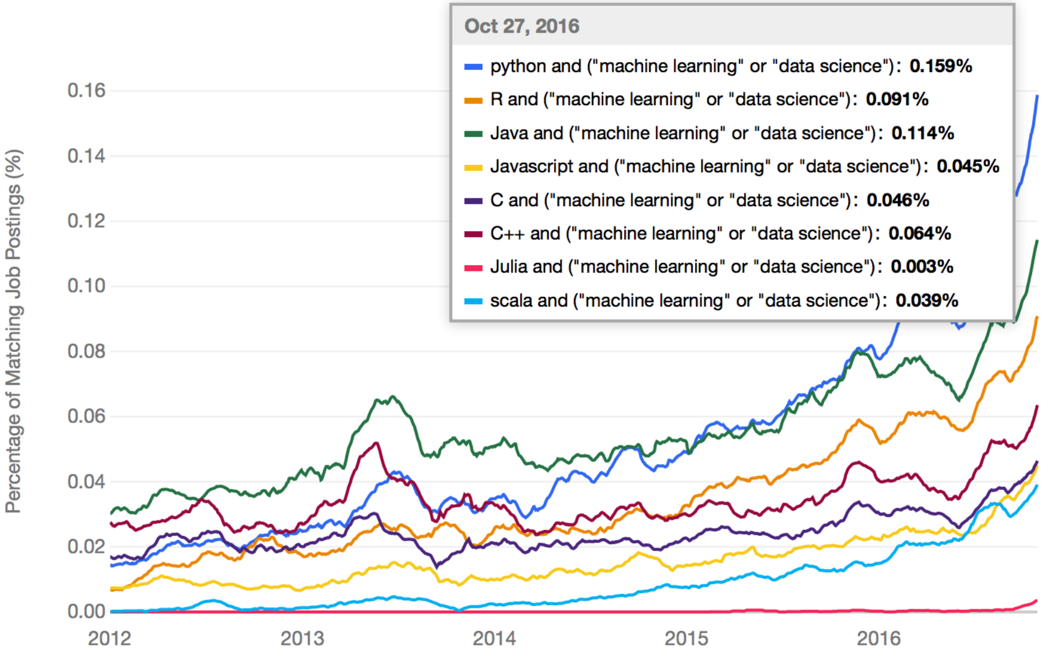
What kinds of problems does Python help solve?

- **Data analytics.**
- **Machine learning and artificial intelligence.**
- Robot path planning.
  - http://shop.oreilly.com/product/0636920024736.do
  - https://www.packtpub.com/application-development/learning-robotics-using-python
- Computational geometry.
  - https://www.youtube.com/watch?v=nb3GRgtjlTw
- Finite Elements and Computational Fluid Dynamics (CFD).
  - http://lorenabarba.com/blog/cfd-python-12-steps-to-navier-stokes/
- Prototyping work for lower-level programming languages (embedded systems development).

Why use Python in Manufacturing?

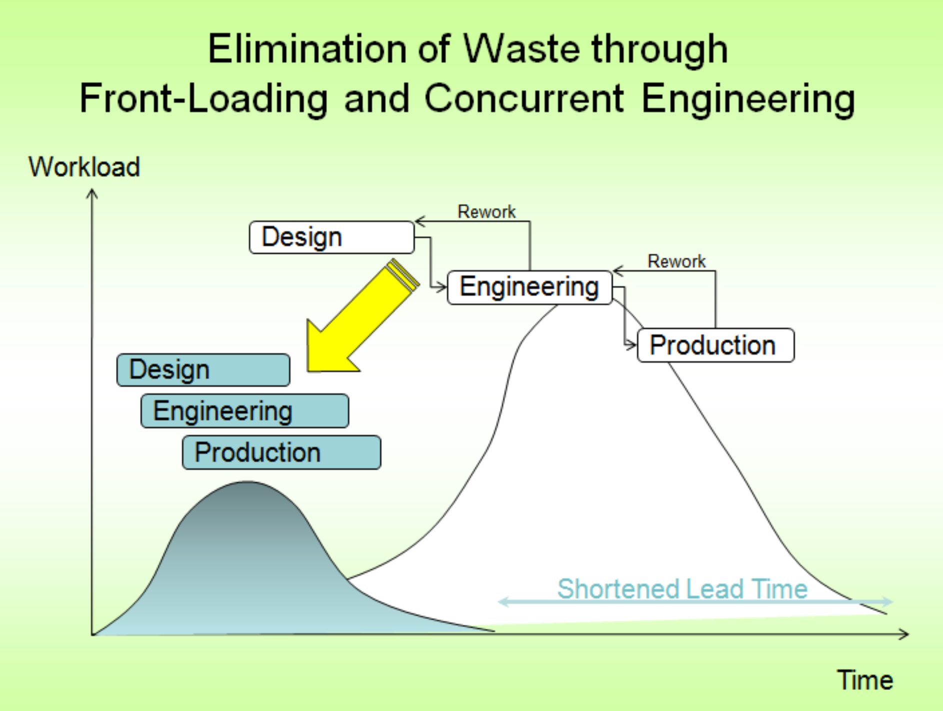
- Python is fast becoming one of the most popular languages in data analytics and machine learning and complex manufacturing processes are producing more data than ever.

  Source: https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en



Why use Python in Engineering?

- Product design is becoming more complex. Engineering silos must be challenged. Mechanical engineers can no longer be ignorant of how software integrates into a system. Complex digital design tooling must be deployed to reduce late-stage design changes.
  Source: https://manufacturingwisdom.com/tag/eating-sequence/



Today’s Agenda

- A brief tour of Python’s interpreter, syntax, libraries and programming basics.

  Reference: Introduction to Python by Jessica McKellar
  http://shop.oreilly.com/product/110000448.do

- A student “project” to check your understanding.

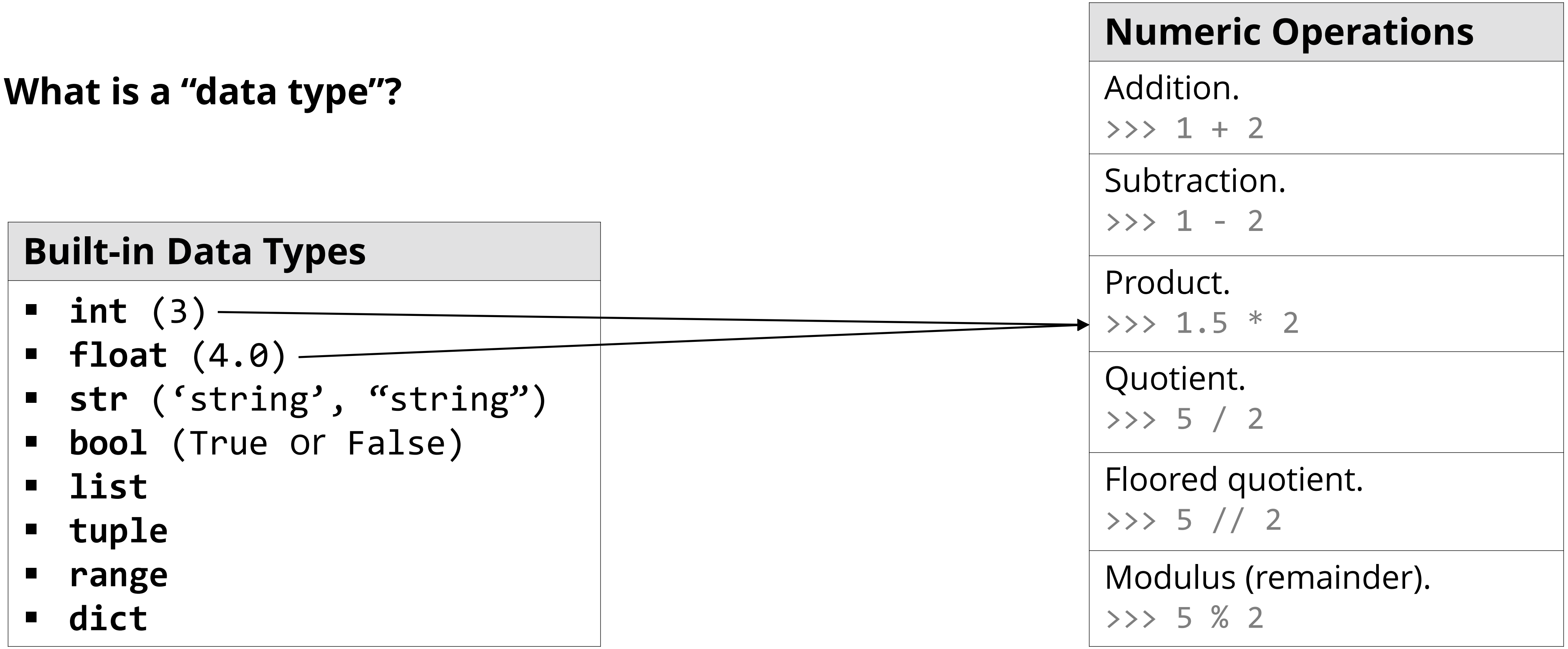
Caveats and Warnings

- Programming is challenging – the following presentation will not make you into an expert. Practice and read code.
- Python is very feature-rich programming language – this presentation will not touch on all of it (or a majority of it).
- We will be looking mostly at imperative code (some OOP), but Python supports multiple paradigms.
- **The Python interpreter is your friend.**
- Execution efficiency and good design principles have been sacrificed for code clarity where possible.
- “Premature optimization is the root of all evil.” – Donald Knuth
- For data analytics and machine learning applications, in particular, knowing Python is not enough.
- Python is generally not suitable for real-time applications which require hard deadlines.

Language Tour

- https://docs.python.org/3/library/functions.html
- https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

What is a “data type”?



Built-in Data Types

- `int (3)`
- `float (4.0)`
- `str (‘string’, “string”)`
- `bool (True or False)`
- `list`
- `tuple`
- `range`
- `dict`

Numeric Operations

Addition.

```
>>> 1 + 2
```

Subtraction.

```
>>> 1 - 2
```

Product.

```
>>> 1.5 * 2
```

Quotient.

```
>>> 5 / 2
```

Floored quotient.

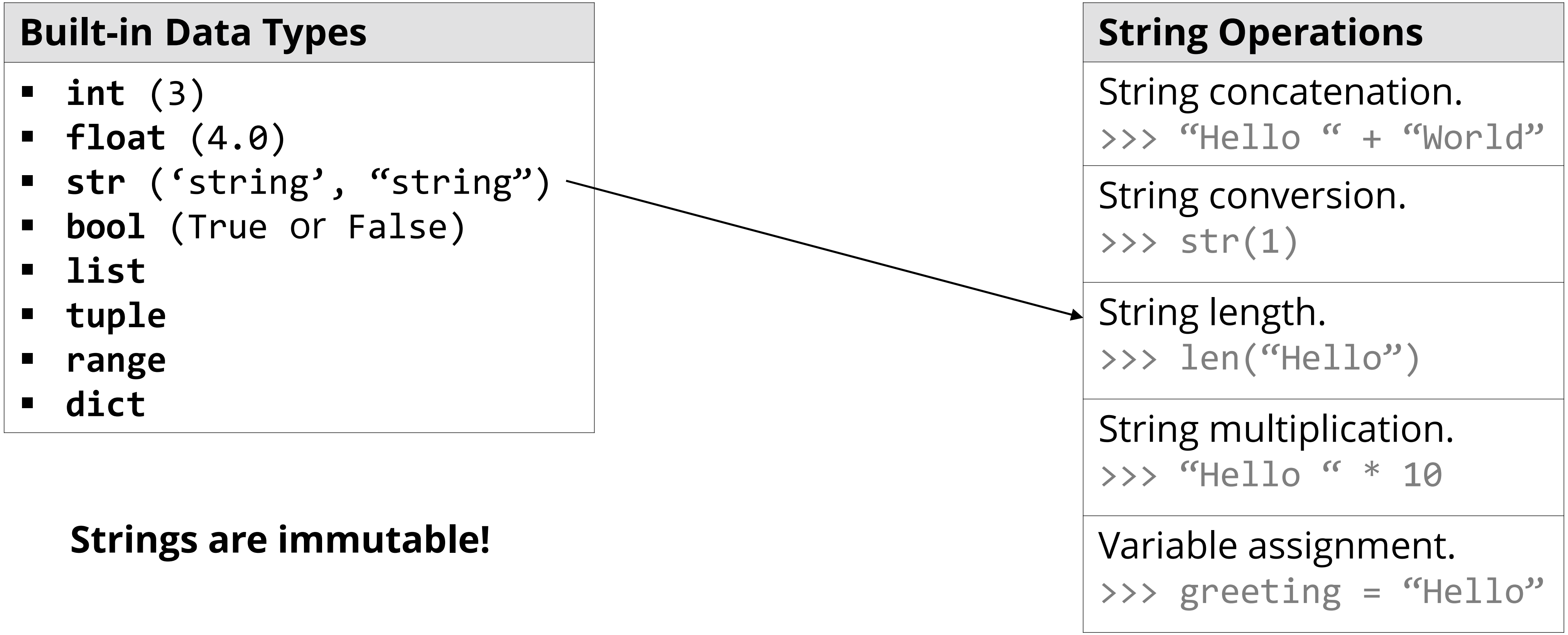
```
>>> 5 // 2
```

Modulus (remainder).

```
>>> 5 % 2
```

Language Tour

- https://docs.python.org/3/library/functions.html
- https://docs.python.org/3/library/stdtypes.html#str



Built-in Data Types

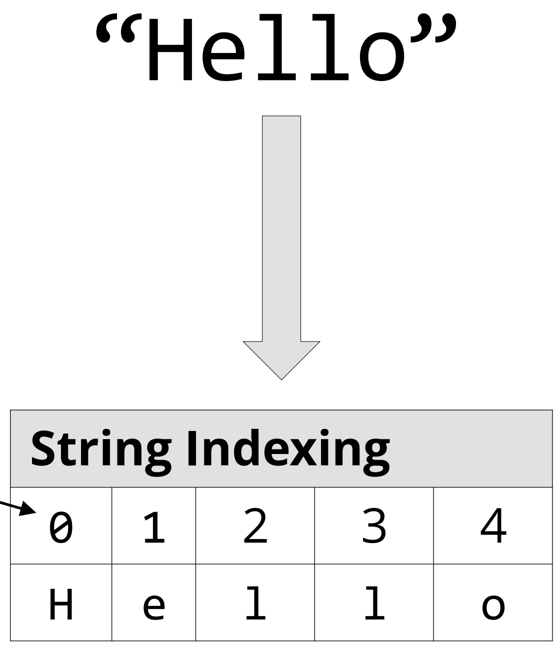
- `int (3)`
- `float (4.0)`
- `str (‘string’, “string”)`
- `bool (True or False)`
- `list`
- `tuple`
- `range`
- `dict`

String Operations
String concatenation. `>>> “Hello “ + “World”`
String conversion. `>>> str(1)`
String length. `>>> len(“Hello”)`
String multiplication. `>>> “Hello “ * 10`
Variable assignment. `>>> greeting = “Hello”`

Strings are immutable!

Language Tour

Note that the index starts from 0. Python used a “0-based” index.



“Hello”

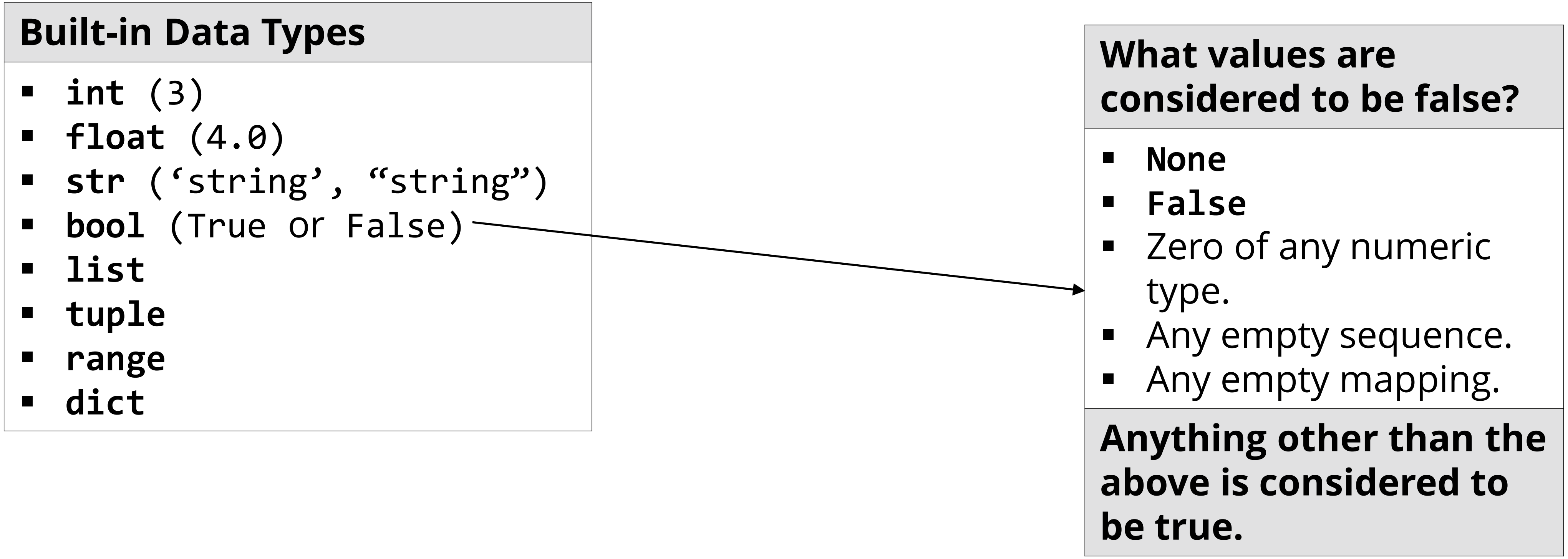
String Indexing				
0	1	2	3	4
H	e	l	l	o

Language Tour

```
>>> a_string = “Hello”
# What is the output of the following operation?
>>> a_string * 10
# What is the output after the following
# operation?
>>> a_string[0]
# What is a_string after the following
# operation?
>>> a_string[0] = “J”
# What is the output of the following
# operations?
>>> a_string[0:3]
>>> a_string[:3]
>>> a_string[1:2]
# What is a_string after the following
# operation?
>>> a_string = “Python”
# Why?
```

Language Tour

- https://docs.python.org/3/library/stdtypes.html#truth-value-testing



Built-in Data Types

- `int (3)`
- `float (4.0)`
- `str (‘string’, “string”)`
- `bool (True or False)`
- `list`
- `tuple`
- `range`
- `dict`

What values are considered to be false?

- `None`
- `False`
- Zero of any numeric type.
- Any empty sequence.
- Any empty mapping.

Anything other than the above is considered to be true.

Language Tour

- https://docs.python.org/3/library/stdtypes.html#boolean-operations-and-or-not

>>> x or y		
x	y	Result
1	1	1
1	0	1
0	1	1
0	0	0

>>> x and y		
x	y	Result
1	1	1
1	0	0
0	1	0
0	0	0

>>> not x	
x	Result
1	0
0	1

Language Tour

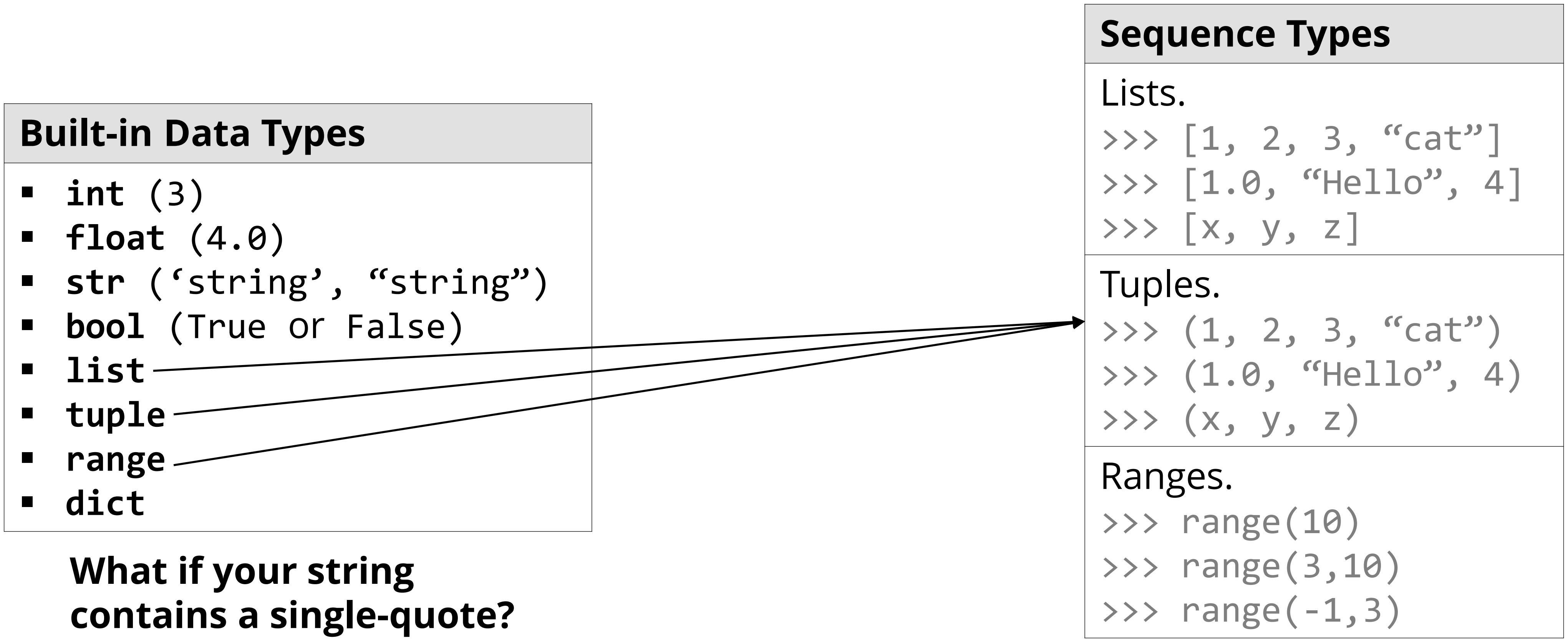
- https://docs.python.org/3/library/stdtypes.html#comparisons

All of these expressions are true.

Comparisons
Less than. `>>> 1 < 2`
Less than or equal to. `>>> 1 <= 1`
Greater than. `>>> 2 > 1.0`
Greater than or equal to. `>>> 2.0 >= 2`
Equality. `>>> 2.0 == 2`
Not equal. `>>> 2.0 != 1`

Language Tour

- https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range



Built-in Data Types

- **int** (3)
- **float** (4.0)
- **str** (‘string’, “string”)
- **bool** (True or False)
- **list**
- **tuple**
- **range**
- **dict**

Sequence Types

Lists.

```
>>> [1, 2, 3, “cat”]
>>> [1.0, “Hello”, 4]
>>> [x, y, z]
```

Tuples.

```
>>> (1, 2, 3, “cat”)
>>> (1.0, “Hello”, 4)
>>> (x, y, z)
```

Ranges.

```
>>> range(10)
>>> range(3,10)
>>> range(-1,3)
```

What if your string contains a single-quote? What about a double-quote?

Language Tour

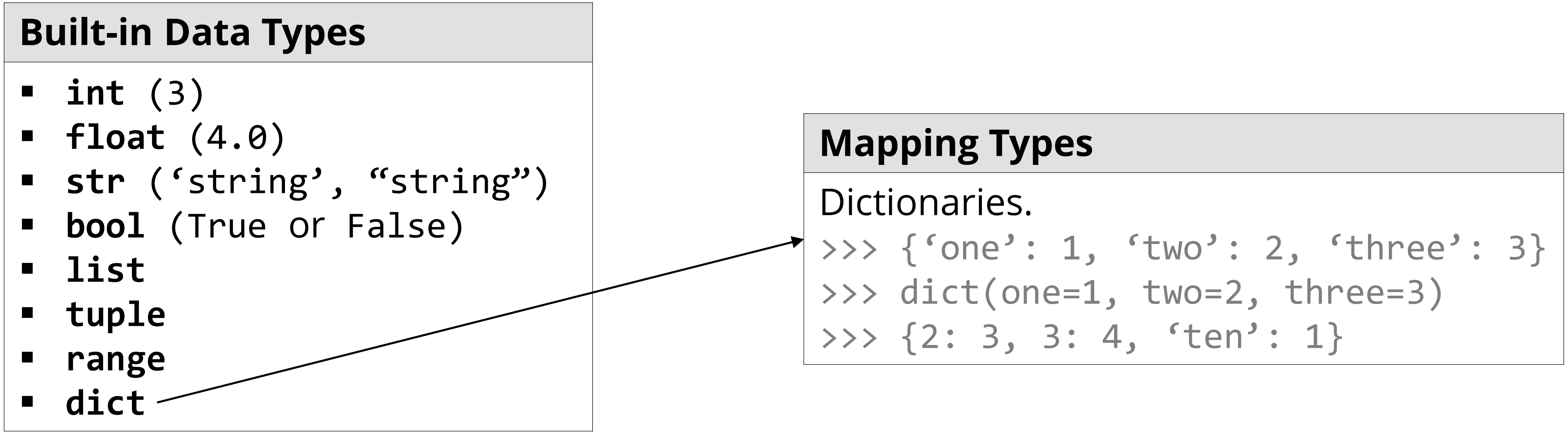
- Lists versus Tuples – what is the **real** difference besides the syntax?
- **Lists are mutable. Tuples are immutable.**
- Lists are often used to store homogenous items. Tuples are typically good for heterogeneous data (2-tuples). Another good use of tuples is when an immutable set of homogenous data is needed (i.e. Cartesian coordinates).
- Tuples are generally faster than lists (allocation and iteration).
- Tuples can be used as dictionary (dict) keys.

Language Tour

```
>>> a_list = [1, 2, 3, 4]
>>> a_tuple = (1, 2, 3, 4)
# What will a_list be after the following
# operation?
>>> a_list[0] = 5
# What will a_tuple be after the following
# operation?
>>> a_tuple[0] = 5
# What is the output of the following
# operations?
>>> len(a_list)
>>> len(a_tuple)
# What is the output of the following
# operations?
>>> range(10)
>>> range(-1, 10)
# What is the output of the following
# operations?
>>> a_list[:3]
>>> a_list[1:3]
>>> a_tuple[:3]
# Which of the three operations above modified
# a_list or a_tuple?
```

Language Tour

- https://docs.python.org/3/library/stdtypes.html#mapping-types-dict



Built-in Data Types

- **int** (3)
- **float** (4.0)
- **str** (‘string’, “string”)
- **bool** (True or False)
- **list**
- **tuple**
- **range**
- **dict**

Mapping Types

Dictionaries.

```
>>> {‘one’: 1, ‘two’: 2, ‘three’: 3}
>>> dict(one=1, two=2, three=3)
>>> {2: 3, 3: 4, ‘ten’: 1}
```

Dictionaries are mutable!

Language Tour

key **value**

```
>>> a_dict = {‘one’: 1, ‘two’: 2, ‘three’: 3}
# What is the output of the following operation?
>>> a_dict[0]
# What is the output of the following operation?
>>> a_dict[‘one’]
# What will a_dict be after the following
# operation?
>>> a_dict[‘two’] = “two”
```

Language Tour

- So, how do we add items to a list? To a dictionary? How you remove an item? All of these methods are hard to keep in your memory.
- We have all of these types flying around…how do you know the type of a variable?
- The answer? Introspection!

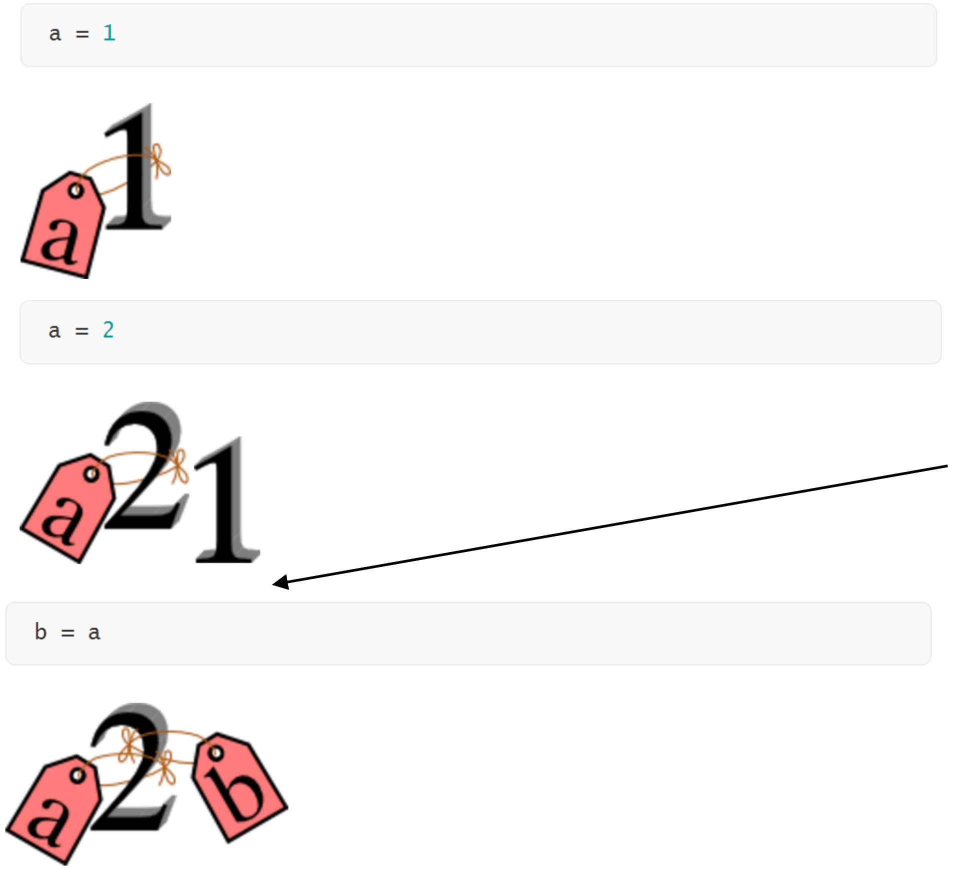
```
# The following print the attributes of the
# provided object.
>>> dir(list)
>>> dir(dict)
>>> dir(“Hello”)
>>> age = 50
# What is the output of the following operation?
>>> type(age)
```

Language Tour

```
good_variable_name = 1
Poorvariablenamechoice = 3.0
3illegal_variable_name = “Hello”
```

- Variables in Python should generally only contain lowercase letters, numbers and underscores. A number **cannot** be used as the first character.
- Avoid abbreviations or industry jargon.
- Python “constants” should use uppercase letters, numbers and underscores.
- When working on existing codebases, **always** write code according to its established conventions.

Language Tour



```
a = 1
```

```
a = 2
```

```
b = a
```

1 is garbage collected here.

- **Source:** http://foobarnbaz.com/2012/07/08/understanding-python-variables/
- How does the Python Garbage Collection (GC) work? See https://www.quora.com/How-does-garbage-collection-in-Python-work/answer/John-Wang-28?srid=mPvm

FOR PUBLIC RELEASE

Language Tour

```
# This is a comment.
```

- Python comments begin the line with a hash (#).
- The comment can extend to the end of a physical line. Multi-line comments must use a hash on each line.
- Comments can be very useful, but it is better to write clear, understandable and *self-documenting* code.
- If you change code, do not let comments fall out-of-date.
- “Code Tells You How, Comments Tell You Why” (https://blog.codinghorror.com/code-tells-you-how-comments-tell-you-why/)

Language Tour

- https://docs.python.org/3/tutorial/inputoutput.html

```
# The ‘print’ function will write output to the console.
print(‘Welcome to this Python event.’)
print(“Found the number”, num)
print(‘There are {} basis vectors in 2D Euclidean space’.format(2))
print(‘The value of pi is {0:.3f}’.format(math.pi))


# The ‘input’ function will receive input from the console.
name = input(‘What is your name? ‘)
```

Language Tour

- Python uses indents (not curly braces) and a colon (:) to establish code blocks.
- Code formatting should, in general, conform to PEP 8.
- When working on existing codebases, **always** write code according to its established conventions.

```
>>> some_value = 42
>>> if some_value == 42:
...     print(‘The value 42 was found.’)
...
>>>
```

suite

4 spaces is convention, no tabs

Language Tour

- https://docs.python.org/3/tutorial/controlflow.html

```
>>> x = int(input(“Please enter an integer: “))
> Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print(‘Negative changed to zero.’)
... elif x == 0:
...     print(‘Zero’)
... elif x == 1:
...     print(‘Single’)
... else:
...     print(‘More’)
...
> More
>>>
```

Language Tour

- To iterate over a sequence, Python provides a **`for`** statement.
- A **`break`** statement in a **`for`** loop immediately sends the control out of the loop.
- A **`continue`** statement in a **`for`** loop immediately returns to the top of the "closest" loop.

```
>>> for i in range(5):
...     print(i)
...
>>>
```

Language Tour

- https://docs.python.org/3/tutorial/controlflow.html

outer loop

inner loop

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             # Why are we doing floor division below?
...             print(n, ‘equals’, x, ‘*’, n//x)
...             # What would happen if the break below was omitted?
...             break
...     else:
...         print(n, ‘is a prime number’)
...
> ???
>>>
```

Language Tour

- https://docs.python.org/3/tutorial/controlflow.html

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print(“Found an even number”, num)
...         # What would happen if the continue below was omitted?
...         continue
...     print(“Found a number”, num)
...
> ???
>>>
```

Language Tour

- https://docs.python.org/3/reference/compound_stmts.html#while

```
>>> counter = 0
>>> while (counter < 10):
...     print(‘The count is:’, count)
...     # What would happen if the line below was omitted?
...     counter = counter + 1
...
> ???
>>>
```

Language Tour

- https://docs.python.org/3/reference/compound_stmts.html#while

```
>>> while (True):
...     number = input(“Enter a number (or ‘q’ to exit): “)
...     if (number == ‘q’):
...         print(“Goodbye!”)
...         break
...     else:
...         print(‘You entered’, number)
...
> ???
>>>
```

Language Tour

- https://docs.python.org/3.3/reference/compound_stmts.html#function-definitions

```
>>> def perform_task():
...     print(“This is a function with no parameters.”)
...
>>>
>>> perform_task() # The function is called here.
```

Function naming convention the same as for variables. But try to use action names!

```
>>> def perform_task(x, y):
...     print(“The sum of x and y is”, x + y)
...
>>>
>>> perform_task() # What does this output?
>>> perform_task(1, 2)
```

Language Tour

- https://docs.python.org/3.3/reference/compound_stmts.html#function-definitions

```
>>> def perform_task(x, y=1):
...     print(“The sum of x and y is”, x + y)
...
>>>
>>> perform_task(4) # What does this output?
```

```
>>> # Is the following function declaration valid? Why or why not?
>>> def perform_task(x=1, y):
...     print(“The sum of x and y is”, x + y)
...
>>>
```

Language Tour

```
>>> z = 2
>>> def perform_task():
...     z = 1
...     print(z)
...
>>> perform_task()
>>> # What will be the output of the following operation? Why?
>>> print(z)
```

```
>>> z = 2
>>> def perform_task():
...     global z
...     z = 1
...     print(z)
...
>>> perform_task()
>>> # What will be the output of the following operation? Why?
>>> print(z)
```

Avoid this at all costs!

Language Tour

- How can we return data from a function?

```
>>> def add(x, y):
...     sum = x + y
...     return sum
...
>>> sum = add(1, 2)
>>> # What will be the value of ‘sum’ here?
```

Language Tour

- How do we use functionality that is not “built-in”? By importing what we need.

```
>>> import random
>>> dir(random)
>>> random.randint(1, 6)
> ???
>>> random.randint(1, 6)
> ???
```

Language Tour

- What is a class? A class is a way to encapsulate data and behavior.
- A class can inherit data and behavior from other classes (subclass inherits from superclass).
- An object is an instance of a class. An object is created when a class is instantiated. A class is the “blueprint” of a plane. The object is the actual, physical plane.

Language Tour

- What is a class? A class is a way to encapsulate data and behavior.
- A class can inherit data and behavior from other classes (subclass inherits from superclass).
- An object is an instance of a class. An object is created when a class is instantiated. A class is the “blueprint” of a plane. The object is the actual, physical plane.
- **All** classes inherit from the **`object`** class.
- **Be careful of deep inheritance levels and multiple inheritance in particular.**

Language Tour

What does defining and using a class look like? See ‘class_demo.py’.

Student Project

Area Calculator Specifications

- Write the program in a Python script (.py file).
- Allow the user to select one of three (3) different geometries – a square, a rectangle or a circle. Allow the user to immediately quit the program by entering ‘q’.
- **Tip: If you are going to use classes, do not use class inheritance. You do not need it.**
- Allow the user to input the appropriate dimensions of the selected geometry.
- Once all dimensions are entered, display the computed area to the user. Print the computed area to two decimal places.
- Allow the user to select the same or a different geometry again or ‘q’ to quit the program.

Further Study Suggestions

- Exception handling.
- Generators (yield).
- Tasks and coroutines.
- Decorators.
- Metaprogramming.
- Python design patterns.
- **Testing** (see ‘Resources’ slide).
- Python data structures (see ‘Resources’ slide).

Application-Specific Suggestions

- Want to build web applications in Python? Check out Django.
- Need a powerful environment for data and geometry visualizations? Check out the Jupyter Project.
- Want to do numerical analysis or linear algebra? Check out SciPy.
- Need to work with deep learning? See Pytorch and/or Theano.
- Need to build a GUI application? See Tkinter.
- Want to build a simulator? Check out Pygame.
- **Any of the above interest you for the next chapter event?**

Resources

Books

- **Matthes, E. (2016). *Python crash course: a hands-on, project-based introduction to programming.* San Francisco: No Starch Press.**
- **Lee, K. D., & Hubbard, S. (2015). *Data Structures and Algorithms with Python.* Cham: Springer International Publishing.**
- **Percival, H. (2014). *Test-driven development with Python.* O'Reilly.**
- Kiusalaas, J. (2010). *Numerical methods in engineering with Python, Second Edition*. Cambridge University Press.
- Solem, J. E. (2012). *Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images.*
- Raschka, S. (2015). *Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics.* Birmingham (U.K.): Packt Publishing.
- VanderPlas, J. (2017). *Python data science handbook: Essential tools for working with data.* Sebastopol, CA: O'Reilly.
- Klein, P. N. (2013). *Coding the matrix: linear algebra through applications to computer science.* Newton, MA: Newtonian Press.

Resources

Videos

- Sarah Guido - Hands-on Data Analysis with Python - PyCon 2015
- Jake VanderPlas - Machine Learning with Scikit-Learn (I) - PyCon 2015
- Olivier Grisel - Machine Learning with Scikit-Learn (II) - PyCon 2015
- **Jessica McKellar: A hands-on introduction to Python for beginning programmers - PyCon 2014**
- Programming Foundations with Python on Udacity

Websites

- Stack Overflow
- Python 3 API Documentation
- The Zen of Python
- Robot Operating System
- Open Source Computer Vision (OpenCV)
- SciPy
- scikit-learn
- **Awesome Python on GitHub**

Thank you!

Thanks for attending!

Special thanks to our hosting partners - Purdue University Northwest and the Commercialization and Manufacturing Excellence Center.

Adam J. Cook

Chief Technical Officer of Alliedstrand

Chair of SME Chapter 112

adam.j.cook@alliedstrand.com

https://twitter.com/AdamJosephCook

https://linkedin.com/in/adam-j-cook

https://github.com/adamjcook

SME

www.sme.org

https://facebook.com/SMEmfg

https://twitter.com/SME_MFG

https://linkedin.com/company/sme

SME Chapter 112

Serving Northwest Indiana and Chicagoland

https://twitter.com/sme112

https://facebook.com/sme112

https://github.com/sme112