Python for Data Science

BLUEGRANITE

2-Day Itinerary

Day 1	Day 2
Python Overview • Executing Commands • Understanding Data Types • Performing Common Operations • Visualization	Advanced Analytics with Python • Object-Oriented Programming • Machine Learning

3-Day Itinerary

Day 1	Day 2	Day 3
Python Overview • Executing Commands • Understanding Data Types • Performing Common Operations	Advanced Analytics with Python • Object-Oriented Programming • Visualization • Machine Learning (part 1)	Advanced Analytics cont'd • Machine Learning (part 2)

About Python

"Python is powerful... and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open."

- Used in:
  - Web and Internet Development
  - Database Access
  - Desktop GUIs
  - Scientific & Numeric
  - Education
  - Network Programming
  - Software & Game Development

- Designed by Guido van Rossum in 1991
- Can be written as Object-Oriented or Functional/Procedural
- Two maintained version streams:
  - Python 3: 3.6
  - Python 2: 2.7

Python 2.x vs. 3.x – Which should I learn?

Python 2.x

- Legacy version of the language
- End-of-life release in 2010
- More extensive set of packages (especially in specialized areas)
- `print "something"`
    - Statement

Python 3.x

- Present and future of the language
- Released in 2008
- Most common packages are available
- `print("something")`
    - Function

Common Packages from SciPy.org

NumPy

Base N-dimensional array package

SciPy library

Fundamental library for scientific computing

Matplotlib

Comprehensive 2D Plotting

IPython

Enhanced Interactive Console

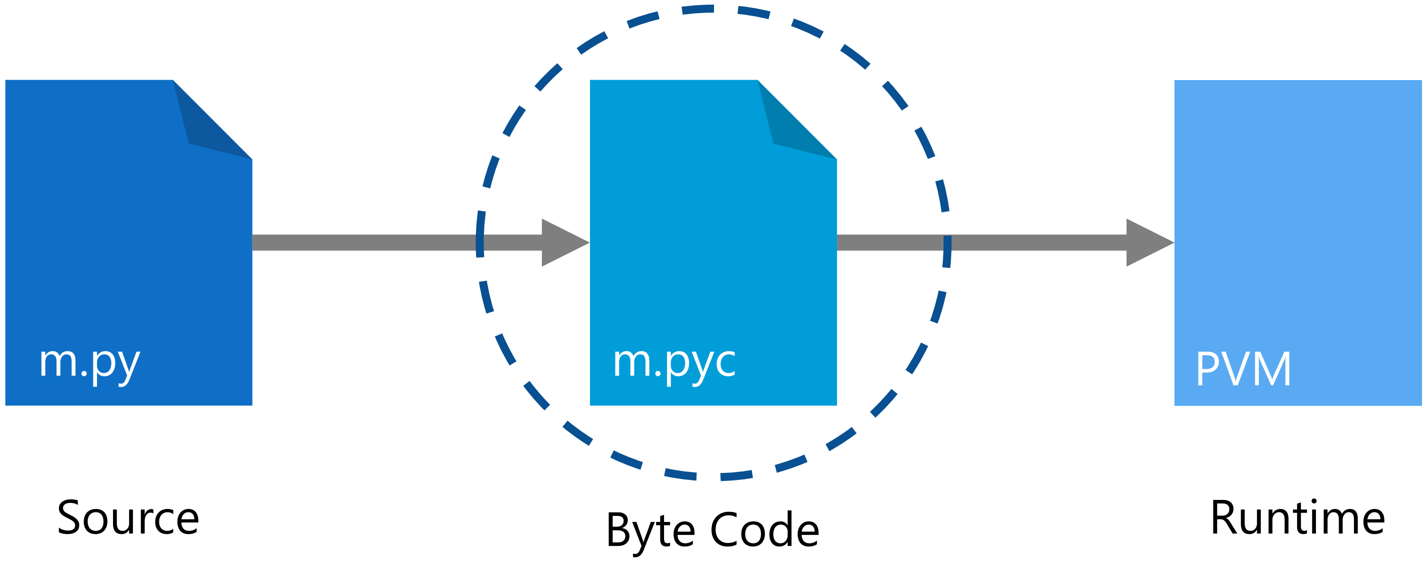
Sympy

Symbolic mathematics

pandas

Data structures & analysis

Traditional Runtime Execution Model



Run Python Interactively

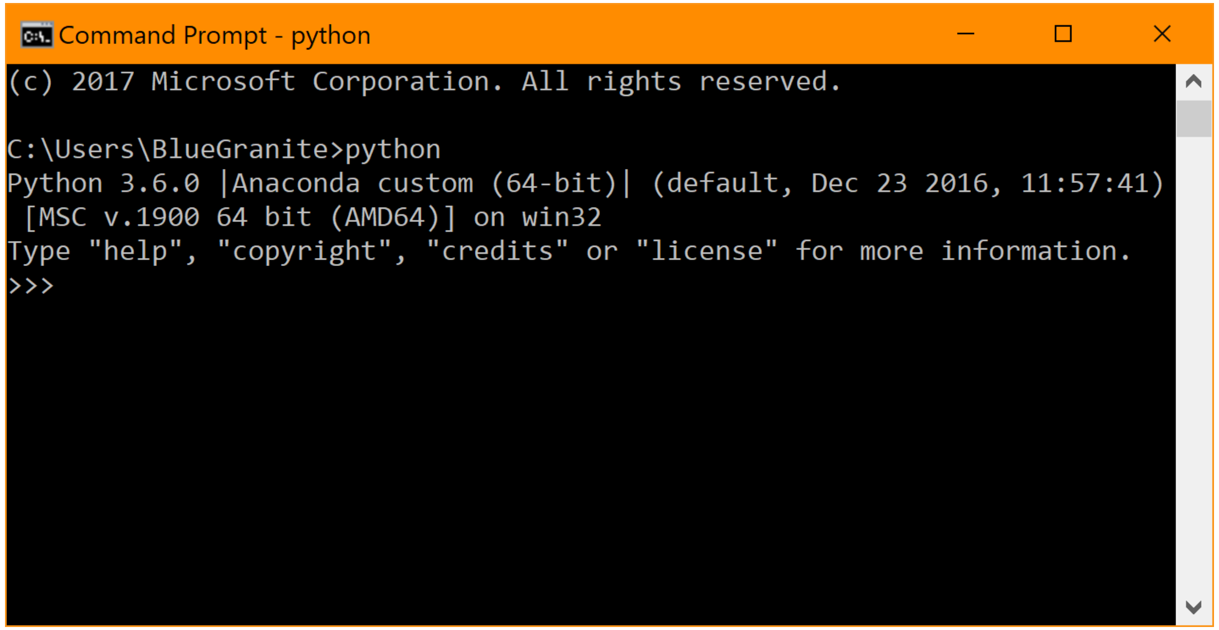
Linux and Mac:

  - Open Terminal

Windows:

  - Open Command Prompt

- Type `python`
  - If you have Python 2 installed as well, you may have to type `python3`
  - Check your python version by typing `python -v`
- To exit interactive mode, type `exit()`



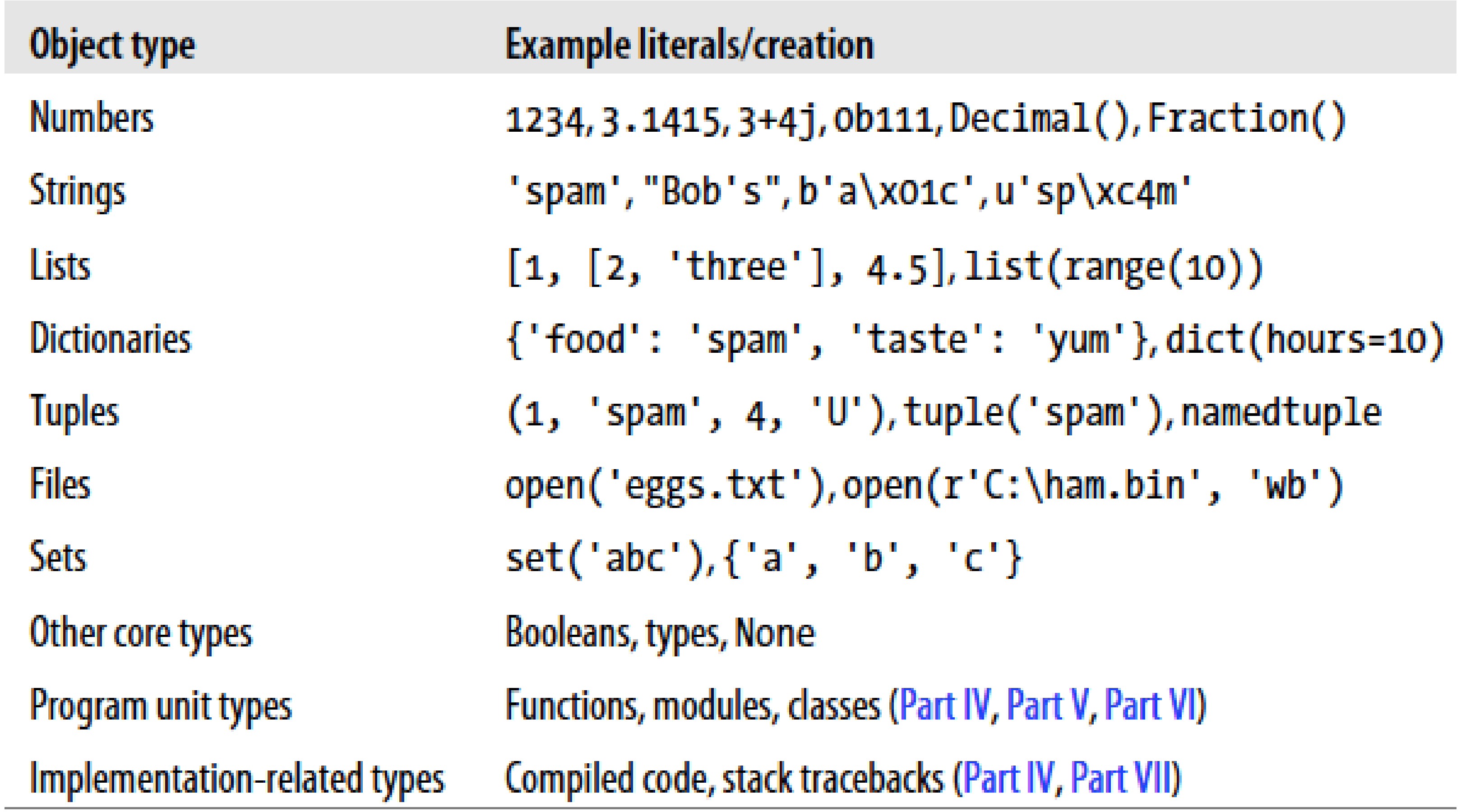
```
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\BlueGranite>python
Python 3.6.0 |Anaconda custom (64-bit)| (default, Dec 23 2016, 11:57:41)
 [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Exploring Python’s Core Data Types

BLUEGRANITE

Python’s Core Data Type



Object type	Example literals/creation
Numbers	`1234`, `3.1415`, `3+4j`, `0b111`, `Decimal()`, `Fraction()`
Strings	`'spam'`, `"Bob's"`, `b'a\x01c'`, `u'sp\xc4m'`
Lists	`[1, [2, 'three'], 4.5]`, `list(range(10))`
Dictionaries	`{'food': 'spam', 'taste': 'yum'}`, `dict(hours=10)`
Tuples	`(1, 'spam', 4, 'U')`, `tuple('spam')`, `namedtuple`
Files	`open('eggs.txt')`, `open(r'C:\ham.bin', 'wb')`
Sets	`set('abc')`, `{'a', 'b', 'c'}`
Other core types	Booleans, types, None
Program unit types	Functions, modules, classes (Part IV, Part V, Part VI)
Implementation-related types	Compiled code, stack tracebacks (Part IV, Part VII)

Numbers

- Number data types only hold numeric values
- Types:
  - Integers
    - Positive or negative whole numbers without a decimal point
  - Floating-point numbers
    - Real numbers and numbers in scientific notation
  - Complex numbers
    - Floats with Imaginary Components
- Immutable

Addition	`5 + 20`
Multiplication	`5 * 5`
Exponentiation	`5 ** 2`

Import math
Use more advanced operations and constants with the 'math' module
	`import math`
	`math.pi`

	`math.sqrt(625) #equivalent to 625 ** 0.5`

Import random
Generate random numbers
	`import random`
	`random.random()`

	`random.choice([1,2,3,4,5])`

Strings

- One of the most popular types in Python.
- Python treats double quotes as the same as single quotes
- Immutable

Assignment	`S = 'Spam'`
Concatenation	`S + 'sy'`
Repetition	`S * 2`
Ranged Slicing	`S[1:3]`
Substringing	`S.find('pa')`
Replacement	`S.replace('pa','li')`
Content Tests	`S.isalpha()`
Splitting and Stripping	`line.split(',')`

Lists

- The most basic data structure
- Starting index is 0
- Uses square brackets: [ ]
- Can house mixed types of objects
- Mutable

Assignment	L = [123, 'spam', 1.23]
Length	len(L)
Indexing	L[0]
Append	L.append('NI')
Pop	L.pop(2)
Reverse	L.reverse()
Sort	L.sort()
Nesting	M = [[1,2,3], [4,5,6], [7,8,9]]

Dictionaries

- A set of key : value pairs
  - Keys must be unique, immutable value
  - Values can be non-unique and of any type.
- Uses curly braces: { }
- Mutable

Assignment	`D = {'food':'Spam',`
	`'quantity':4,`
	`'color':'pink'}`

Finding Values	`D['food']`
Changing Values	`D['quantity'] += 1`

Nesting	`rec['jobs'].append('janitor')`
	`rec['jobs'].remove('mgr')`

Sets

- Recent addition to Python.
- Neither mappings nor sequences.
- Unordered collections of unique and immutable objects.
  - Great for filtering out duplicates or determining differences.
- Uses curly braces: { }
- Mutable

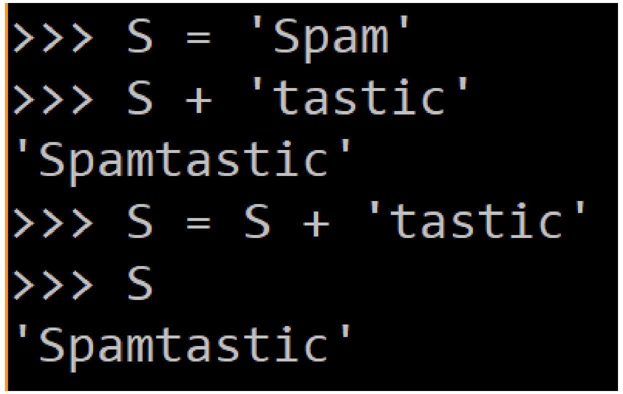
Assignment	`X = set('spam')`
	`Y = {'h', 'a', 'm'}`

Intersection	`X & Y`
Union	`X | Y`
Difference	`X – Y`
Superset	`X > Y`
Comparison	`X == Y`

What is Mutability?

Immutable

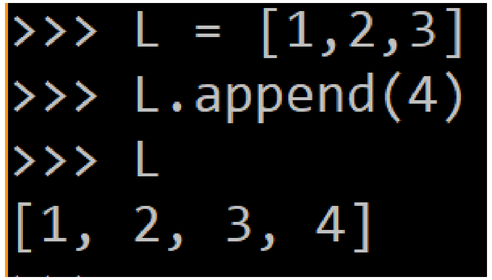
- Unchangeable without reassignment
- Core types:
  - Numbers, Strings, and Tuples



```
>>> S = 'Spam'
>>> S + 'tastic'
'Spamtastic'
>>> S = S + 'tastic'
>>> S
'Spamtastic'
```

Mutable

- Can be directly changed with a function
- Core types:
  - Lists, Dictionaries, and Sets

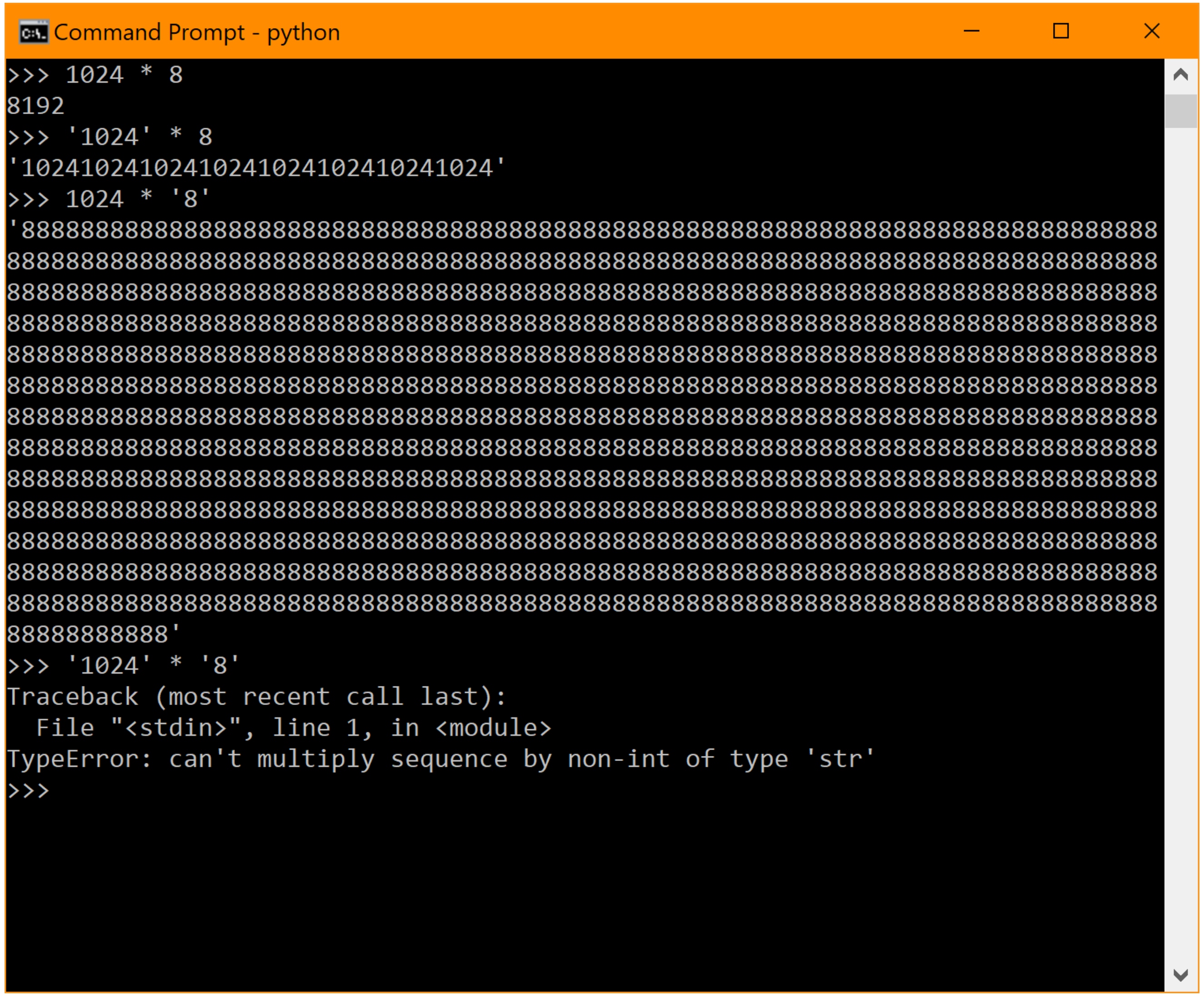


```
>>> L = [1,2,3]
>>> L.append(4)
>>> L
[1, 2, 3, 4]
```

Polymorphisms

In Python, operations can apply to variables regardless of their type.

However, a single operator behaves differently given the variable type.



```
>>> 1024 * 8
8192
>>> '1024' * 8
'10241024102410241024102410241024'
>>> 1024 * '8'
'88888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
888888888888888888888888888888888888888888888888888888888888888888888888888888
88888888888'
>>> '1024' * '8'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
>>>
```

If Statements

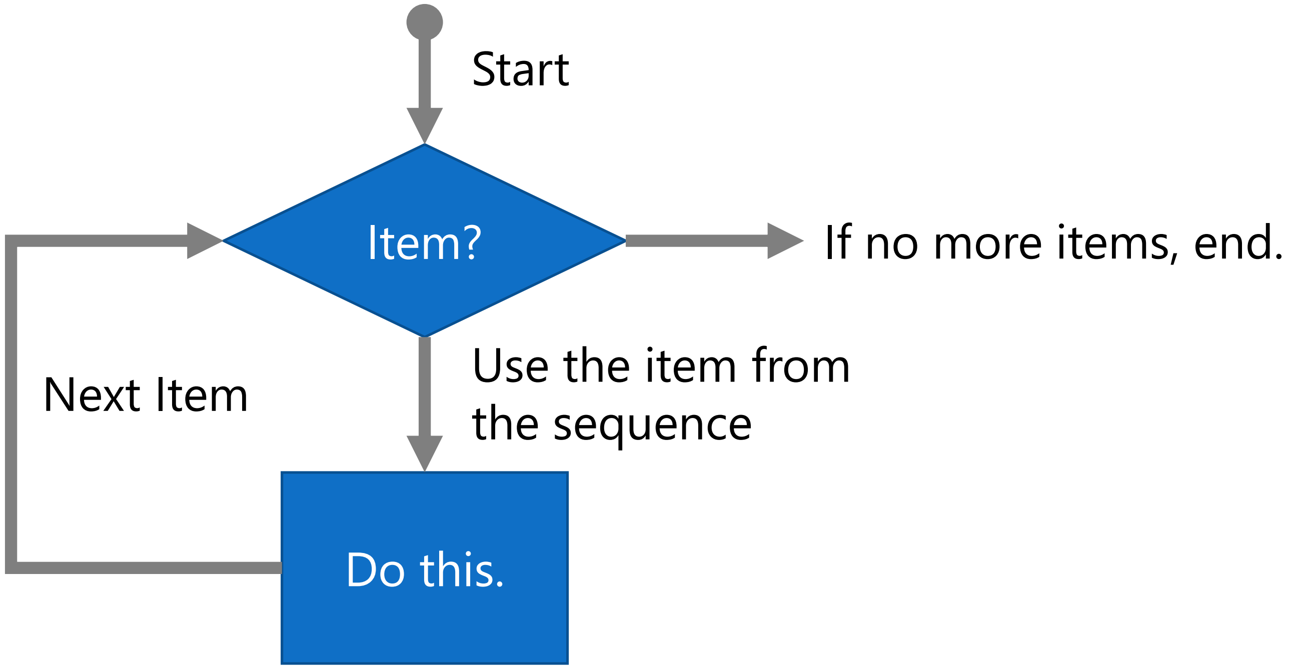
- Conditional statements
  - Useful for changing the operations of your code by different situations.

```
x = 5
if x > 5:
    print('x is > 5')
elif x < 5:
    print('x is < 5')
else:
    print('x is 5')
```

For Loops

```
J = 'lumberjack'
for j in J:
    print(j,end='-')
```

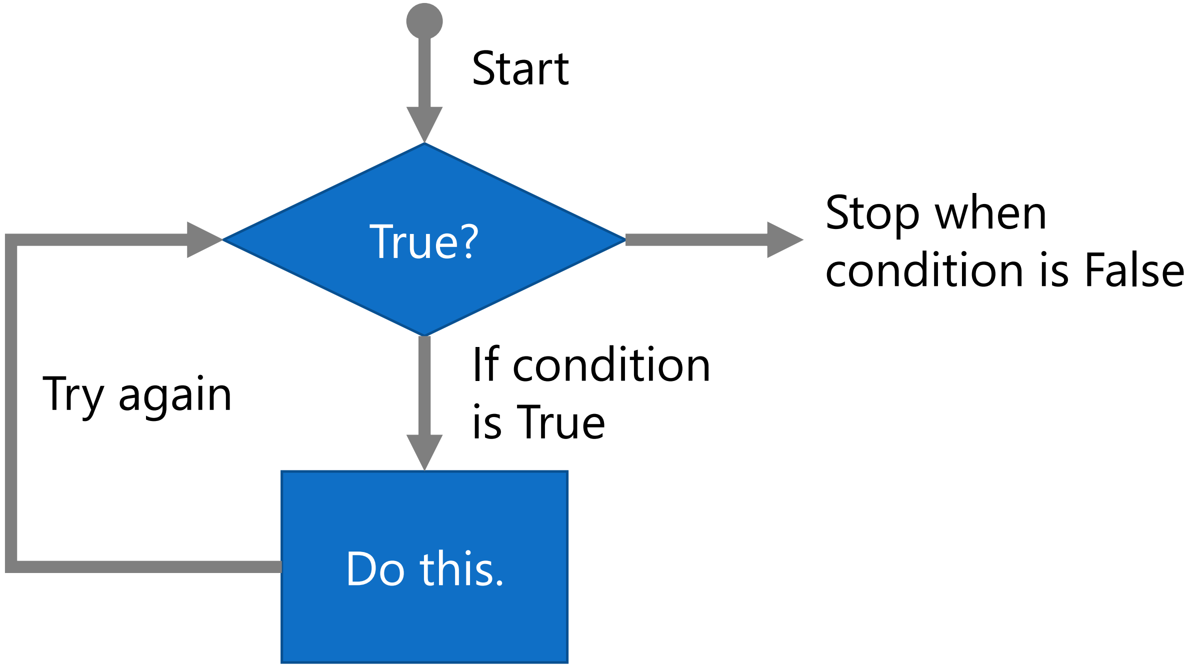
- Indexed loop
  - Uses an index to cycle through the code and perform some operation.
  - Useful when operations need to be completed repetitively over a set.
  - Will increase computational complexity



While Loops

- Conditional loop
  - Uses a condition to know when to cycle through the code and perform some operation.
  - Useful when operations need to be completed repetitively only until a condition is met.
  - Will increase computational complexity

```
a=0; b=10
while a<b:
    print(a,end=' ')
    a += 1
```



Breaks and Continues

Breaks

- Jumps out of the closest enclosing loops (past the entire loop statement)
  - Causes an immediate exit from a loop

```
while True:
    name = input('Enter name: ')
    if name == 'stop': break
    age = input('Enter age: ')

print('Hello',name,'=>',int(age)
**2)
```

Continues

- Jumps to the top of the closest enclosing loop.
  - Returns to the header line of the loop

```
x = 10
while x:
    x -= 1
    if x % 2 != 0: continue
#If odd, skip print
    print(x,end=' ')
```

Define your own Functions

- The packages you use everyday are simply collections of code in functions that allow for repeatable operations.
- You can create your own by simply using the `def` statement.

def times(x,y):	#Name the function and define the arguments
return x * y	#Tell the function what to return

times(2,4)	#Try out the times function
Output: 8	

times('Cat',5)	#Functions are typeless
Output: 'CatCatCatCatCat'	

Install Packages

To install packages in Python, you will need to use a secondary tool.

- BlueGranite recommends *Anaconda*, but Python.org recommends *pip*.

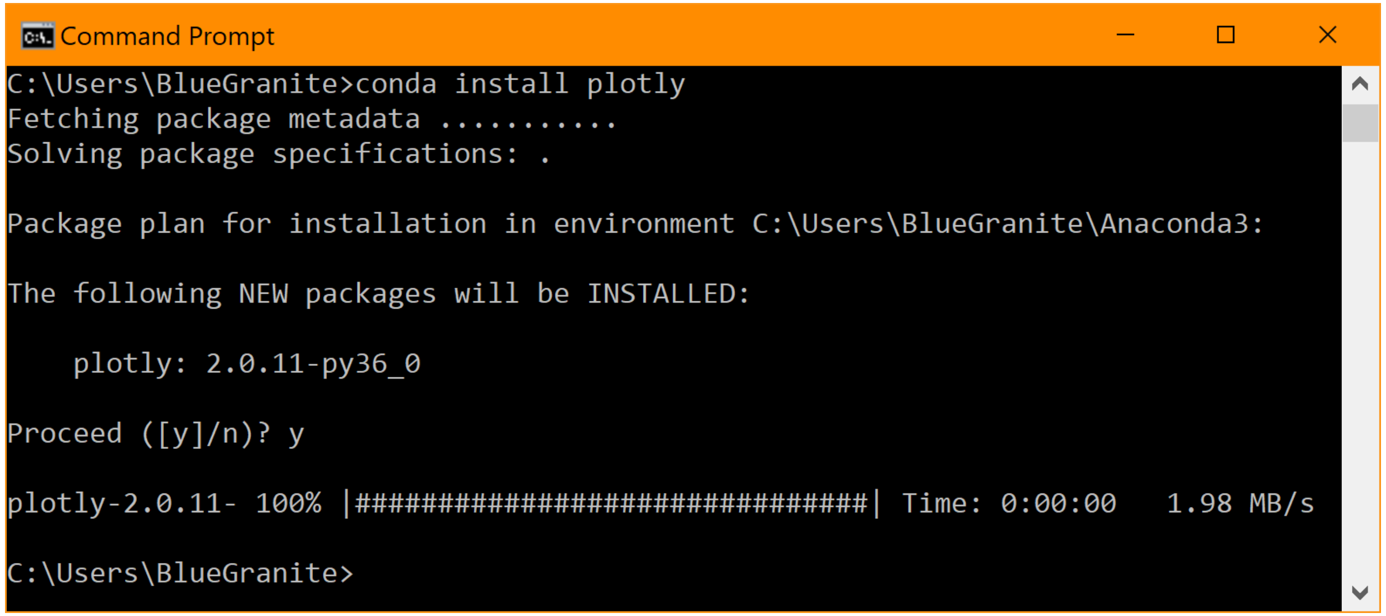
Python uses PyPl, the Python Package Library as the repository for packages.

Anaconda:

- Type `conda install <packagename>`

Pip and SetupTools:

- Type `pip install <packagename>`



```
C:\Users\BlueGranite>conda install plotly
Fetching package metadata ...........
Solving package specifications: .

Package plan for installation in environment C:\Users\BlueGranite\Anaconda3:

The following NEW packages will be INSTALLED:

    plotly: 2.0.11-py36_0

Proceed ([y]/n)? y

plotly-2.0.11- 100% |###############################| Time: 0:00:00   1.98 MB/s

C:\Users\BlueGranite>
```

Visualization

BLUEGRANITE

Visualization Packages for Python

There are many visualization packages for Python, but some of the most popular are:

- Matplotlib
- Seaborn
- Bokeh
- Plot.ly

Machine Learning

BLUEGRANITE

Machine Learning Packages for Python

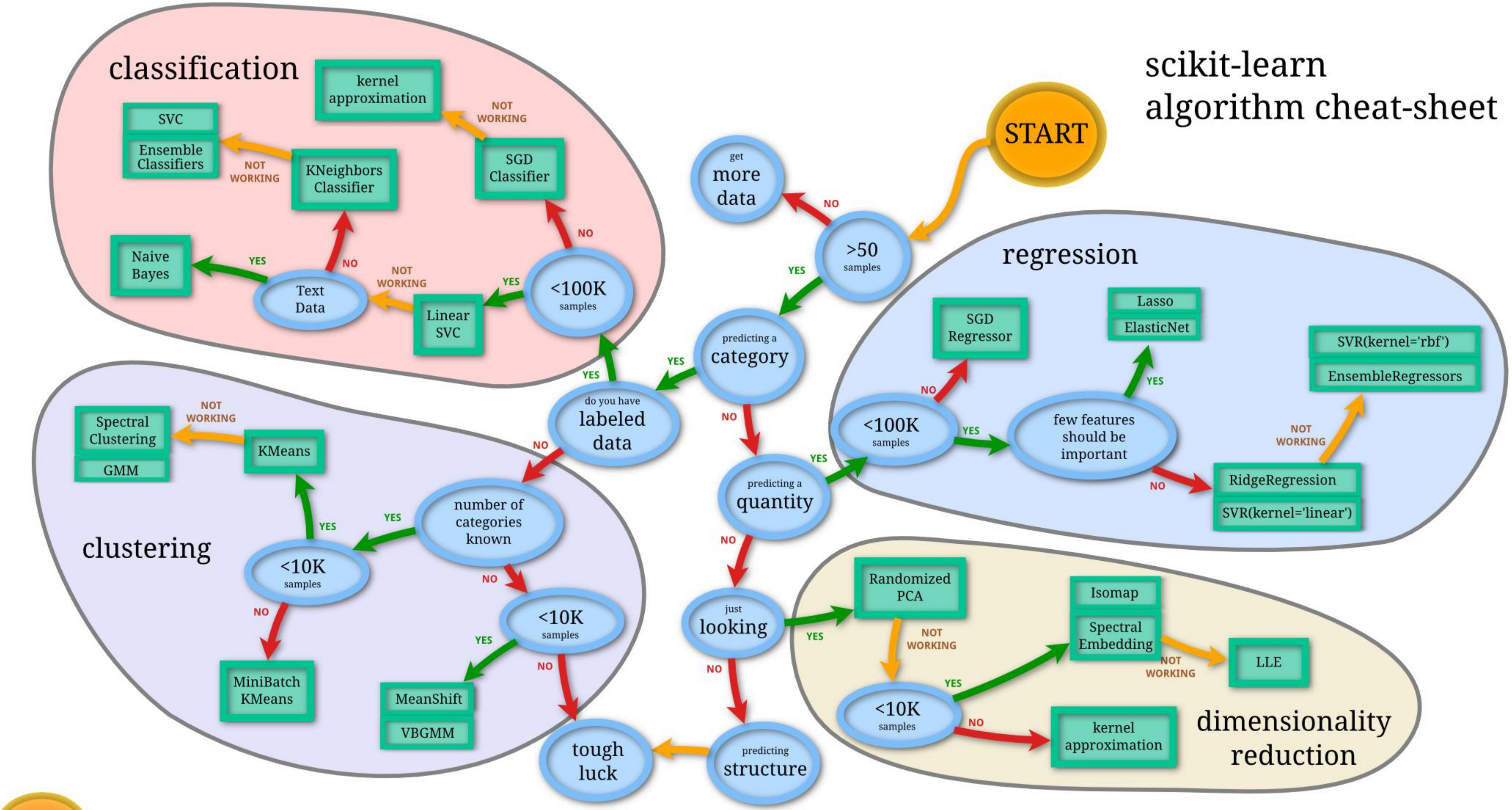
Scikit-Learn

The most popular machine learning package for Python.

- Includes:
  - Generalized Linear Models
  - Discriminant Analyses
  - Support Vector Machines
  - Gradient Descent
  - Nearest Neighbors
  - K-Means
  - Decision Trees
  - Neural Networks
  - …and more!

Others Notable Packages

- Useful for Neural Networks:
  - TensorFlow
  - Theano
  - Pylearn2
  - Pyevolve
- NLP
  - Pattern
- Computer Vision
  - Caffe
  - OpenCV



scikit-learn algorithm cheat-sheet

START

>50 samples

NO: get more data

YES: predicting a category

predicting a category, YES: do you have labeled data

do you have labeled data, YES: classification

predicting a category, NO: predicting a quantity

predicting a quantity, YES: regression

predicting a quantity, NO: just looking

just looking, YES: dimensionality reduction

just looking, NO: predicting structure

predicting structure: tough luck

classification

<100K samples

NO: SGD Classifier

NOT WORKING: kernel approximation

YES: Linear SVC

NOT WORKING: Text Data

YES: Naive Bayes

NO: KNeighbors Classifier

NOT WORKING: SVC / Ensemble Classifiers

clustering

do you have labeled data, NO: number of categories known

YES: <10K samples

YES: KMeans

NOT WORKING: Spectral Clustering / GMM

NO: MiniBatch KMeans

number of categories known, NO: <10K samples

YES: MeanShift / VBGMM

NO: tough luck

regression

<100K samples

NO: SGD Regressor

YES: few features should be important

YES: Lasso / ElasticNet

NO: RidgeRegression / SVR(kernel='linear')

NOT WORKING: SVR(kernel='rbf') / EnsembleRegressors

dimensionality reduction

Randomized PCA

NOT WORKING: <10K samples

YES: Isomap / Spectral Embedding

NOT WORKING: LLE

NO: kernel approximation

Back