finxter

The Ultimate Python Cheat Sheet

Keywords

Keyword	Description	Code Examples
`False`, `True`	Boolean data type	`False == (1 > 2)` `True == (2 > 1)`

`and`, `or`, `not`	Logical operators
	→ Both are true
	→ Either is true
	→ Flips Boolean
		`True and True  # True`
		`True or False  # True`
		`not False  # True`

`break`	Ends loop prematurely	`while True:` `  break # finite loop`

`continue`	Finishes current loop iteration	`while True:`
		`  continue`
		`  print("42") # dead code`

`class`	Defines new class	`class Coffee:` `  # Define your class`
`def`	Defines a new function or class method.	`def say_hi():` `  print('hi')`

`if`, `elif`, `else`	Conditional execution:
	- "if" condition == True?
	- "elif" condition == True?
	- Fallback: else branch
		`x = int(input("ur val:"))`
		`if  x > 3:  print("Big")`
		`elif x == 3:  print("3")`
		`else:  print("Small")`

`for`, `while`	`# For loop`
	`for i in [0,1,2]:`
	`  print(i)`
		`# While loop does same`
		`j = 0`
		`while j < 3:`
		`  print(j); j = j + 1`

`in`	Sequence membership	`42 in [2, 39, 42] # True`

`is`	Same object memory location	`y = x = 3`
		`x is y  # True`
		`[3] is [3]  # False`

`None`	Empty value constant	`print() is None # True`
`lambda`	Anonymous function	`(lambda x: x+3)(3) # 6`

`return`	Terminates function. Optional return value defines function result.	`def increment(x):`
		`  return x + 1`
		`increment(4) # returns 5`

Basic Data Structures

Type	Description	Code Examples

Boolean	The Boolean data type is either `True` or `False`. Boolean operators are ordered by priority: `not` → `and` → `or`
	{ } → 👎
	{1, 2, 3} → 👍
		`## Evaluates to True:`
		`1<2 and 0<=1 and 3>2 and 2>=2 and 1==1 and 1!=0`
		`## Evaluates to False:`
		`bool(None or 0 or 0.0 or '' or [] or {} or set())`
		Rule: *None, 0, 0.0, empty strings, or empty container types evaluate to False*

Integer, Float	An **integer** is a positive or negative number without decimal point such as `3`.
	A **float** is a positive or negative number with floating point precision such as `3.1415926`.
	Integer division rounds toward the smaller integer (example: `3//2==1`).
		`## Arithmetic Operations`
		`x, y = 3, 2`
		`print(x + y)  # = 5`
		`print(x - y)  # = 1`
		`print(x * y)  # = 6`
		`print(x / y)  # = 1.5`
		`print(x // y)  # = 1`
		`print(x % y)  # = 1`
		`print(-x)  # = -3`
		`print(abs(-x))  # = 3`
		`print(int(3.9))  # = 3`
		`print(float(3))  # = 3.0`
		`print(x ** y)  # = 9`

String	Python Strings are sequences of characters.
	String Creation Methods:
	1. Single quotes
	`>>> 'Yes'`
	2. Double quotes
	`>>> "Yes"`
	3. Triple quotes (multi-line)
	`>>> """Yes`
	`  We Can"""`
	4. String method
	`>>> str(5) == '5'`
	`True`
	5. Concatenation
	`>>> "Ma" + "hatma"`
	`'Mahatma'`
	Whitespace chars:
	Newline `\n`,
	Space `\s`,
	Tab `\t`
		`## Indexing and Slicing`
		`s = "The youngest pope was 11 years"`
		`s[0] # 'T'`
		`s[1:3] # 'he'`
		`s[-3:-1] # 'ar'`
		`s[-3:] # 'ars'`
		Slice [::2]
		0 1 2 3
		1 2 3 4
		`x = s.split()`
		`x[-2] + " " + x[2] + "s" # '11 popes'`
		`## String Methods`
		`y = "  Hello world\t\n  "`
		`y.strip() # Remove Whitespace`
		`"HI".lower() # Lowercase: 'hi'`
		`"hi".upper() # Uppercase: 'HI'`
		`"hello".startswith("he") # True`
		`"hello".endswith("lo") # True`
		`"hello".find("ll") # Match at 2`
		`"cheat".replace("ch", "m") # 'meat'`
		`''.join(["F", "B", "I"]) # 'FBI'`
		`len("hello world") # Length: 15`
		`"ear" in "earth" # True`

Complex Data Structures

Type	Description	Example
List	Stores a sequence of elements. Unlike strings, you can modify list objects (they're *mutable*).	`l = [1, 2, 2]` `print(len(l)) # 3`

Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation.	`[1, 2].append(4) # [1, 2, 4]`
		`[1, 4].insert(1,9) # [1, 9, 4]`
		`[1, 2] + [4] # [1, 2, 4]`

Removal	Slow for lists	`[1, 2, 2, 4].remove(1) # [2, 2, 4]`
Reversing	Reverses list order	`[1, 2, 3].reverse() # [3, 2, 1]`
Sorting	Sorts list using fast Timsort	`[2, 4, 2].sort() # [2, 2, 4]`

Indexing	Finds the first occurrence of an element & returns index. Slow worst case for whole list traversal.	`[2, 2, 4].index(2)`
		`# index of item 2 is 0`
		`[2, 2, 4].index(2,1)`
		`# index of item 2 after pos 1 is 1`

Stack	Use Python lists via the list operations append() and pop()	`stack = [3]`
		`stack.append(42) # [3, 42]`
		`stack.pop() # 42 (stack: [3])`
		`stack.pop() # 3 (stack: [])`

Set	An unordered collection of unique elements (*at-most-once*) → fast membership *O(1)*	`basket = {'apple', 'eggs',`
		`  'banana', 'orange'}`
		`same = set(['apple', 'eggs',`
		`  'banana', 'orange'])`

Dictionary	Useful data structure for storing (key, value) pairs	`cal = {'apple' : 52, 'banana' : 89,` `  'choco' : 546} # calories`

Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the **`keys()`** and **`values()`** functions to access all keys and values of the dictionary	`print(cal['apple'] < cal['choco'])`
		`# True`
		`cal['cappu'] = 74`
		`print(cal['banana'] < cal['cappu'])`
		`# False`
		`print('apple' in cal.keys()) # True`
		`print(52 in cal.values())  # True`

Dictionary Iteration	You can access the (key, value) pairs of a dictionary with the **`items()`** method.	`for k, v in cal.items():`
		`  print(k) if v > 500 else ''`
		`# 'choco'`

Membership operator	Check with the **`in`** keyword if set, list, or dictionary contains an element. Set membership is faster than list membership.	`basket = {'apple', 'eggs',`
		`  'banana', 'orange'}`
		`print('eggs' in basket)  # True`
		`print('mushroom' in basket) # False`

List & set comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses. Set comprehension works similar to list comprehension.	`l = ['hi ' + x for x in ['Alice', 'Bob', 'Pete']]`
		`# ['Hi Alice', 'Hi Bob', 'Hi Pete']`
		`l2 = [x * y for x in range(3) for y in range(3) if x>y] # [0, 0, 2]`
		`squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}`

Python Cheat Sheet: Keywords

"A puzzle a day to learn, code, and play" → Visit finxter.com

Keyword	Description	Code example
`False, True`	Data values from the data type Boolean	`False == (1 > 2), True == (2 > 1)`

`and, or, not`	Logical operators:
	`(x and y)` → both x and y must be True
	`(x or y)` → either x or y must be True
	`(not x)` → x must be false
		`x, y = True, False`
		`(x or y) == True  # True`
		`(x and y) == False  # True`
		`(not y) == True  # True`

`break`	Ends loop prematurely	`while(True):`
		`  break # no infinite loop`
		`print("hello world")`

`continue`	Finishes current loop iteration	`while(True):`
		`  continue`
		`  print("43") # dead code`

`class`
 	
`def`
	Defines a new class → a real-world concept (object oriented programming)
	
	Defines a new function or class method. For latter, first parameter ("self") points to the class object. When calling class method, first parameter is implicit.
		`class Beer:`
		`  def __init__(self):`
		`  self.content = 1.0`
		`  def drink(self):`
		`  self.content = 0.0`
		
		`becks = Beer() # constructor - create class`
		`becks.drink() # beer empty: b.content == 0`

`if, elif, else`	Conditional program execution: program starts with "if" branch, tries the "elif" branches, and finishes with "else" branch (until one branch evaluates to True).	`x = int(input("your value: "))`
		`if x > 3: print("Big")`
		`elif x == 3: print("Medium")`
		`else: print("Small")`

`for, while`	`# For loop declaration`
	`for i in [0,1,2]:`
	`  print(i)`
		`# While loop - same semantics`
		`j = 0`
		`while j < 3:`
		`  print(j)`
		`  j = j + 1`

`in`	Checks whether element is in sequence	`42 in [2, 39, 42] # True`

`is`	Checks whether both elements point to the same object	`y = x = 3`
		`x is y # True`
		`[3] is [3] # False`

`None`	Empty value constant	`def f():`
		`  x = 2`
		`f() is None # True`

`lambda`	Function with no name (anonymous function)	`(lambda x: x + 3)(3) # returns 6`

`return`	Terminates execution of the function and passes the flow of execution to the caller. An optional value after the return keyword specifies the function result.	`def incrementor(x):`
		`  return x + 1`
		`incrementor(4) # returns 5`

finxter

Python Cheat Sheet: Basic Data Types

"*A puzzle a day to learn, code, and play*" → Visit finxter.com

	Description	Example

Boolean	The Boolean data type is a truth value, either `True` or `False`.
	
	The Boolean operators ordered by priority:
	`not x` → "if x is False, then x, else y"
	`x and y` → "if x is False, then x, else y"
	`x or y` → "if x is False, then y, else x"
	
	These comparison operators evaluate to `True`:
	`1 < 2 and 0 <= 1 and 3 > 2 and 2 >=2 and 1 == 1 and 1 != 0 # True`
		`## 1. Boolean Operations`
		`x, y = True, False`
		`print(x and not y) # True`
		`print(not x and y or x) # True`
		
		`## 2. If condition evaluates to False`
		`if None or 0 or 0.0 or '' or [] or {} or set():`
		`  # None, 0, 0.0, empty strings, or empty`
		`  # container types are evaluated to False`
		`  print("Dead code") # Not reached`

Integer, Float	An integer is a positive or negative number without floating point (e.g. `3`). A float is a positive or negative number with floating point precision (e.g. `3.14159265359`).
	
	The '`//`' operator performs integer division. The result is an integer value that is rounded toward the smaller integer number (e.g. `3 // 2 == 1`).
		`## 3. Arithmetic Operations`
		`x, y = 3, 2`
		`print(x + y) # = 5`
		`print(x - y) # = 1`
		`print(x * y) # = 6`
		`print(x / y) # = 1.5`
		`print(x // y) # = 1`
		`print(x % y) # = 1s`
		`print(-x) # = -3`
		`print(abs(-x)) # = 3`
		`print(int(3.9)) # = 3`
		`print(float(3)) # = 3.0`
		`print(x ** y) # = 9`

String	Python Strings are sequences of characters.
	
	The four main ways to create strings are the following.
	
	1. Single quotes
	`'Yes'`
	2. Double quotes
	`"Yes"`
	3. Triple quotes (multi-line)
	`"""Yes`
	`We Can"""`
	4. String method
	`str(5) == '5' # True`
	5. Concatenation
	`"Ma" + "hatma" # 'Mahatma'`
	
	These are whitespace characters in strings.
	• Newline `\n`
	• Space `\s`
	• Tab `\t`
		`## 4. Indexing and Slicing`
		`s = "The youngest pope was 11 years old"`
		`print(s[0])  # 'T'`
		`print(s[1:3])  # 'he'`
		`print(s[-3:-1])  # 'ol'`
		`print(s[-3:])  # 'old'`
		`x = s.split()  # creates string array of words`
		`print(x[-3] + " " + x[-1] + " " + x[2] + "s")`
		`  # '11 old popes'`
		
		`## 5. Most Important String Methods`
		`y = "  This is lazy\t\n  "`
		`print(y.strip()) # Remove Whitespace: 'This is lazy'`
		`print("DrDre".lower()) # Lowercase: 'drdre'`
		`print("attention".upper()) # Uppercase: 'ATTENTION'`
		`print("smartphone".startswith("smart")) # True`
		`print("smartphone".endswith("phone")) # True`
		`print("another".find("other")) # Match index: 2`
		`print("cheat".replace("ch", "m")) # 'meat'`
		`print(','.join(["F", "B", "I"])) # 'F,B,I'`
		`print(len("Rumpelstiltskin")) # String length: 15`
		`print("ear" in "earth") # Contains: True`

finxter

Python Cheat Sheet: Complex Data Types

"A puzzle a day to learn, code, and play" → Visit finxter.com

	Description	Example
List	A container data type that stores a sequence of elements. Unlike strings, lists are mutable: modification possible.	`l = [1, 2, 2]` `print(len(l)) # 3`

Adding elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation.
	The append operation is very fast.
		`[1, 2, 2].append(4) # [1, 2, 2, 4]`
		`[1, 2, 4].insert(2,2) # [1, 2, 2, 4]`
		`[1, 2, 2] + [4] # [1, 2, 2, 4]`

Removal	Removing an element can be slower.	`[1, 2, 2, 4].remove(1) # [2, 2, 4]`
Reversing	This reverses the order of list elements.	`[1, 2, 3].reverse() # [3, 2, 1]`
Sorting	Sorts a list. The computational complexity of sorting is linear in the no. list elements.	`[2, 4, 2].sort() # [2, 2, 4]`
Indexing	Finds the first occurence of an element in the list & returns its index. Can be slow as the whole list is traversed.	`[2, 2, 4].index(2) # index of element 4 is "0"` `[2, 2, 4].index(2,1) # index of element 2 after pos 1 is "1"`

Stack	Python lists can be used intuitively as stacks via the two list operations append() and pop().	`stack = [3]`
		`stack.append(42) # [3, 42]`
		`stack.pop() # 42 (stack: [3])`
		`stack.pop() # 3 (stack: [])`

Set	A set is an unordered collection of unique elements ("at-most-once").	`basket = {'apple', 'eggs', 'banana', 'orange'}` `same = set(['apple', 'eggs', 'banana', 'orange'])`
Dictionary	The dictionary is a useful data structure for storing (key, value) pairs.	`calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}`

Reading and writing elements	Read and write elements by specifying the key within the brackets. Use the keys() and values() functions to access all keys and values of the dictionary.	`print(calories['apple'] < calories['choco']) # True`
		`calories['cappu'] = 74`
		`print(calories['banana'] < calories['cappu']) # False`
		`print('apple' in calories.keys()) # True`
		`print(52 in calories.values()) # True`

Dictionary Looping	You can access the (key, value) pairs of a dictionary with the `items()` method.	`for k, v in calories.items():` `  print(k) if v > 500 else None # 'chocolate'`

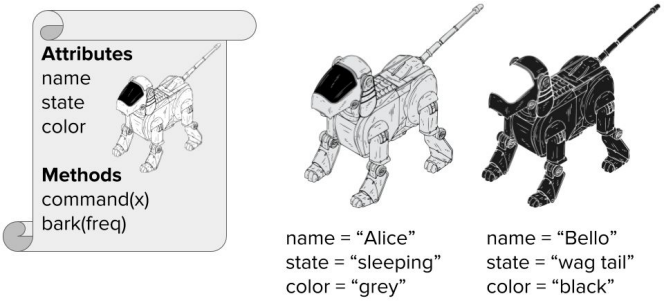
Membership operator	Check with the 'in' keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.	`basket = {'apple', 'eggs', 'banana', 'orange'}`
		`print('eggs' in basket) # True`
		`print('mushroom' in basket) # False`

List and Set Comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses.
	
	Set comprehension is similar to list comprehension.
		`# List comprehension`
		`l = [('Hi ' + x) for x in ['Alice', 'Bob', 'Pete']]`
		`print(l) # ['Hi Alice', 'Hi Bob', 'Hi Pete']`
		`l2 = [x * y for x in range(3) for y in range(3) if x>y]`
		`print(l2) # [0, 0, 2]`
		`# Set comprehension`
		`squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}`

finxter

Python Cheat Sheet: Classes

"A puzzle a day to learn, code, and play" → Visit finxter.com

	Description	Example
Classes	A class encapsulates data and functionality: data as attributes, and functionality as methods. It is a blueprint for creating concrete instances in memory. Class Instances 	
Instance	You are an instance of the class human. An instance is a concrete implementation of a class: all attributes of an instance have a fixed value. Your hair is blond, brown, or black--but never unspecified. Each instance has its own attributes independent of other instances. Yet, class variables are different. These are data values associated with the class, not the instances. Hence, all instance share the same class variable `species` in the example.	
Self	The first argument when defining any method is always the `self` argument. This argument specifies the instance on which you call the method. `self` gives the Python interpreter the information about the concrete instance. To *define* a method, you use `self` to modify the instance attributes. But to *call* an instance method, you do not need to specify `self`.	

Creation	You can create classes "on the fly" and use them as logical units to store complex data types.
	
	`class Employee():`
	`  pass`
	`employee = Employee()`
	`employee.salary = 122000`
	`employee.firstname = "alice"`
	`employee.lastname = "wonderland"`
	
	`print(employee.firstname + " "`
	`  + employee.lastname + " "`
	`  + str(employee.salary) + "$")`
	`# alice wonderland 122000$`	

```
class Dog:
        """ Blueprint of a dog """

        # class variable shared by all instances
        species = ["canis lupus"]

        def __init__(self, name, color):
                self.name = name
                self.state = "sleeping"
                self.color = color

        def command(self, x):
                if x == self.name:
                        self.bark(2)
                elif x == "sit":
                        self.state = "sit"
                else:
                        self.state = "wag tail"

        def bark(self, freq):
                for i in range(freq):
                        print("[" + self.name
                                + "]: Woof!")

bello = Dog("bello", "black")
alice = Dog("alice", "white")

print(bello.color) # black
print(alice.color) # white

bello.bark(1) # [bello]: Woof!

alice.command("sit")
print("[alice]: " + alice.state)
# [alice]: sit

bello.command("no")
print("[bello]: " + bello.state)
# [bello]: wag tail

alice.command("alice")
# [alice]: Woof!
# [alice]: Woof!

bello.species += ["wulf"]
print(len(bello.species)
        == len(alice.species)) # True (!)
```

finxter

Python Cheat Sheet: Functions and Tricks

"A puzzle a day to learn, code, and play" → Visit finxter.com

		Description	Example	Result
ADVANCED FUNCTIONS	`map(func, iter)`	Executes the function on all elements of the iterable	`list(map(lambda x: x[0], ['red', 'green', 'blue']))`	`['r', 'g', 'b']`
	`map(func, i1, ..., ik)`	Executes the function on all k elements of the k iterables	`list(map(lambda x, y: str(x) + ' ' + y + 's' , [0, 2, 2], ['apple', 'orange', 'banana']))`	`['0 apples', '2 oranges', '2 bananas']`
	`string.join(iter)`	Concatenates iterable elements separated by `string`	`' marries '.join(list(['Alice', 'Bob']))`	`'Alice marries Bob'`
	`filter(func, iterable)`	Filters out elements in iterable for which function returns `False` (or 0)	`list(filter(lambda x: True if x>17 else False, [1, 15, 17, 18]))`	`[18]`
	`string.strip()`	Removes leading and trailing whitespaces of string	`print("\n  \t  42  \t ".strip())`	`42`
	`sorted(iter)`	Sorts iterable in ascending order	`sorted([8, 3, 2, 42, 5])`	`[2, 3, 5, 8, 42]`
	`sorted(iter, key=key)`	Sorts according to the key function in ascending order	`sorted([8, 3, 2, 42, 5], key=lambda x: 0 if x==42 else x)`	`[42, 2, 3, 5, 8]`
	`help(func)`	Returns documentation of `func`	`help(str.upper())`	`'... to uppercase.'`
	`zip(i1, i2, ...)`	Groups the i-th elements of iterators `i1, i2, ...` together	`list(zip(['Alice', 'Anna'], ['Bob', 'Jon', 'Frank']))`	`[('Alice', 'Bob'), ('Anna', 'Jon')]`
	Unzip	Equal to: 1) unpack the zipped list, 2) zip the result	`list(zip(*[('Alice', 'Bob'), ('Anna', 'Jon')]))`	`[('Alice', 'Anna'), ('Bob', 'Jon')]`
	`enumerate(iter)`	Assigns a counter value to each element of the iterable	`list(enumerate(['Alice', 'Bob', 'Jon']))`	`[(0, 'Alice'), (1, 'Bob'), (2, 'Jon')]`
TRICKS	python -m http.server <P>	Want to share files between PC and phone? Run this command in PC's shell. <P> is any port number 0–65535. Type <IP address of PC>:<P> in the phone's browser. You can now browse the files in the PC directory.		
	Read comic	`import antigravity`	Open the comic series xkcd in your web browser	
	Zen of Python	`import this`	`'...Beautiful is better than ugly. Explicit is ...'`	
	Swapping numbers	Swapping variables is a breeze in Python. No offense, Java!	`a, b = 'Jane', 'Alice'` `a, b = b, a`	`a = 'Alice'` `b = 'Jane'`

	Unpacking arguments	Use a sequence as function arguments via asterisk operator *. Use a dictionary (key, value) via double asterisk operator **	`def f(x, y, z): return x + y * z`
			`f(*[1, 3, 4])`
			`f(**{'z' : 4, 'x' : 1, 'y' : 3})`
				
				`13`
				`13`

	Extended Unpacking	Use unpacking for multiple assignment feature in Python	`a, *b = [1, 2, 3, 4, 5]`	`a = 1` `b = [2, 3, 4, 5]`

	Merge two dictionaries	Use unpacking to merge two dictionaries into a single one	`x={'Alice' : 18}`
			`y={'Bob' : 27, 'Ann' : 22}`
			`z = {**x,**y}`
				`z = {'Alice': 18, 'Bob': 27, 'Ann': 22}`

finxter

Python Cheat Sheet: 14 Interview Questions

"A puzzle a day to learn, code, and play" → Visit finxter.com

Question	Code	Question	Code

Check if list contains integer x	`l = [3, 3, 4, 5, 2, 111, 5]`
	`print(111 in l) # True`
		Get missing number in [1...100]	`def get_missing_number(lst):`
			`  return set(range(lst[len(lst)-1])[1:]) - set(l)`
			`l = list(range(1,100))`
			`l.remove(50)`
			`print(get_missing_number(l)) # 50`

Find duplicate number in integer list	`def find_duplicates(elements):`
	`  duplicates, seen = set(), set()`
	`  for element in elements:`
	`  if element in seen:`
	`  duplicates.add(element)`
	`  seen.add(element)`
	`  return list(duplicates)`
		Compute the intersection of two lists	`def intersect(lst1, lst2):`
			`  res, lst2_copy = [], lst2[:]`
			`  for el in lst1:`
			`  if el in lst2_copy:`
			`  res.append(el)`
			`  lst2_copy.remove(el)`
			`  return res`

Check if two strings are anagrams	`def is_anagram(s1, s2):`
	`  return set(s1) == set(s2)`
	`print(is_anagram("elvis", "lives")) # True`
		Find max and min in unsorted list	`l = [4, 3, 6, 3, 4, 888, 1, -11, 22, 3]`
			`print(max(l)) # 888`
			`print(min(l)) # -11`

Remove all duplicates from list	`lst = list(range(10)) + list(range(10))`
	`lst = list(set(lst))`
	`print(lst)`
	`# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
		Reverse string using recursion	`def reverse(string):`
			`  if len(string)<=1: return string`
			`  return reverse(string[1:])+string[0]`
			`print(reverse("hello")) # olleh`

Find pairs of integers in list so that their sum is equal to integer x	`def find_pairs(l, x):`
	`  pairs = []`
	`  for (i, el_1) in enumerate(l):`
	`  for (j, el_2) in enumerate(l[i+1:]):`
	`  if el_1 + el_2 == x:`
	`  pairs.append((el_1, el_2))`
	`  return pairs`
		Compute the first n Fibonacci numbers	`a, b = 0, 1`
			`n = 10`
			`for i in range(n):`
			`  print(b)`
			`  a, b = b, a+b`
			`# 1, 1, 2, 3, 5, 8, ...`

Check if a string is a palindrome	`def is_palindrome(phrase):`
	`  return phrase == phrase[::-1]`
	`print(is_palindrome("anna")) # True`
		Sort list with Quicksort algorithm	`def qsort(L):`
			`  if L == []: return []`
			`  return qsort([x for x in L[1:] if x< L[0]]) + L[0:1] +`
			`qsort([x for x in L[1:] if x>=L[0]])`
			`lst = [44, 33, 22, 5, 77, 55, 999]`
			`print(qsort(lst))`
			`# [5, 22, 33, 44, 55, 77, 999]`

Use list as stack, array, and queue	`# as a list ...`
	`l = [3, 4]`
	`l +=  [5, 6] # l = [3, 4, 5, 6]`
	
	`# ... as a stack ...`
	`l.append(10) # l = [4, 5, 6, 10]`
	`l.pop() # l = [4, 5, 6]`
	
	`# ... and as a queue`
	`l.insert(0, 5) # l = [5, 4, 5, 6]`
	`l.pop() # l = [5, 4, 5]`
		Find all permutations of string	`def get_permutations(w):`
			`  if len(w)<=1:`
			`  return set(w)`
			`  smaller = get_permutations(w[1:])`
			`  perms = set()`
			`  for x in smaller:`
			`  for pos in range(0,len(x)+1):`
			`  perm = x[:pos] + w[0] + x[pos:]`
			`  perms.add(perm)`
			`  return perms`
			`print(get_permutations("nan"))`
			`# {'nna', 'ann', 'nan'}`

finxter

Python Cheat Sheet: NumPy

"*A puzzle a day to learn, code, and play*" → Visit finxter.com

Name	Description	Example
`a.shape`	The shape attribute of NumPy array a keeps a tuple of integers. Each integer describes the number of elements of the axis.	`a = np.array([[1,2],[1,1],[0,0]])` `print(np.shape(a))  # (3, 2)`
`a.ndim`	The ndim attribute is equal to the length of the shape tuple.	`print(np.ndim(a))  # 2`

`*`	The asterisk (star) operator performs the Hadamard product, i.e., multiplies two matrices with equal shape element-wise.	`a = np.array([[2, 0], [0, 2]])`
		`b = np.array([[1, 1], [1, 1]])`
		`print(a*b)  # [[2 0] [0 2]]`

`np.matmul(a,b), a@b`	The standard matrix multiplication operator. Equivalent to the @ operator.	`print(np.matmul(a,b))` `# [[2 2] [2 2]]`
`np.arange([start, ]stop, [step, ])`	Creates a new 1D numpy array with evenly spaced values	`print(np.arange(0,10,2))` `# [0 2 4 6 8]`
`np.linspace(start, stop, num=50)`	Creates a new 1D numpy array with evenly spread elements within the given interval	`print(np.linspace(0,10,3))` `# [ 0.  5. 10.]`
`np.average(a)`	Averages over all the values in the numpy array	`a = np.array([[2, 0], [0, 2]])` `print(np.average(a))  # 1.0`

`<slice> = <val>`	Replace the <slice> as selected by the slicing operator with the value <val>.	`a = np.array([0, 1, 0, 0, 0])`
		`a[::2] = 2`
		`print(a)  # [2 1 2 0 2]`

`np.var(a)`	Calculates the variance of a numpy array.	`a = np.array([2, 6])` `print(np.var(a))  # 4.0`
`np.std(a)`	Calculates the standard deviation of a numpy array	`print(np.std(a))  # 2.0`

`np.diff(a)`	Calculates the difference between subsequent values in NumPy array a	`fibs = np.array([0, 1, 1, 2, 3, 5])`
		`print(np.diff(fibs, n=1))`
		`# [1 0 1 1 2]`

`np.cumsum(a)`	Calculates the cumulative sum of the elements in NumPy array a.	`print(np.cumsum(np.arange(5)))` `# [ 0  1  3  6 10]`

`np.sort(a)`	Creates a new NumPy array with the values from a (ascending).	`a = np.array([10,3,7,1,0])`
		`print(np.sort(a))`
		`# [ 0  1  3  7 10]`

`np.argsort(a)`	Returns the indices of a NumPy array so that the indexed values would be sorted.	`a = np.array([10,3,7,1,0])`
		`print(np.argsort(a))`
		`# [4 3 1 2 0]`

`np.max(a)`	Returns the maximal value of NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.max(a))  # 10`
`np.argmax(a)`	Returns the index of the element with maximal value in the NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.argmax(a))  # 0`
`np.nonzero(a)`	Returns the indices of the nonzero elements in NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.nonzero(a))  # [0 1 2 3]`

finxter

finxter Book: Simplicity – The Finer Art of Creating Software

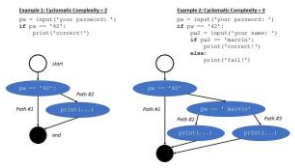
Complexity

"A whole, made up of parts—difficult to analyze, understand, or explain".

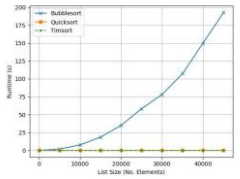
Complexity appears in
- Project Lifecycle
- Code Development
- Algorithmic Theory
- Processes
- Social Networks
- Learning & Your Daily Life

Project Lifecycle

Cyclomatic Complexity



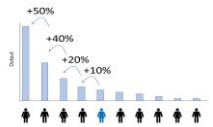
Runtime Complexity

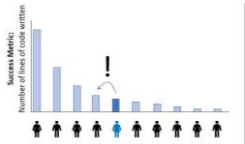


→ Complexity reduces productivity and focus. It'll consume your precious time. Keep it simple!

80/20 Principle

Majority of effects come from the minority of causes.



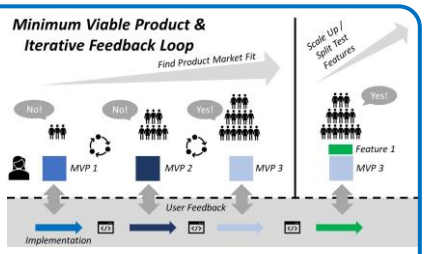


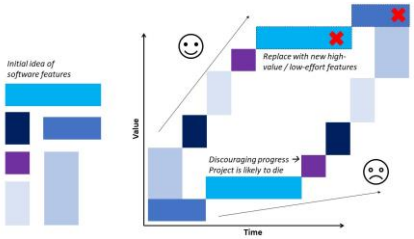
Pareto Tips
1. Figure out your success metrics.
2. Figure out your big goals in life.
3. Look for ways to achieve the same things with fewer resources.
4. Reflect on your own successes
5. Reflect on your own failures
6. Read more books in your industry.
7. Spend much of your time improving and tweaking existing products
8. Smile.
9. Don't do things that reduce value

Maximize Success Metric:
#lines of code written

Minimum Viable Product (MVP)

A minimum viable product in the software sense is code that is stripped from all features to focus on the core functionality.





How to MVP?
- Formulate hypothesis
- Omit needless features
- Split test to validate each new feature
- Focus on product-market fit
- Seek high-value and low-cost features

Clean Code Principles

1. You Ain't Going to Need It
2. The Principle of Least Surprise
3. Don't Repeat Yourself
4. **Code For People Not Machines**
5. Stand on the Shoulders of Giants
6. Use the Right Names
7. Single-Responsibility Principle
8. Use Comments
9. Avoid Unnecessary Comments
10. Be Consistent
11. Test
12. Think in Big Pictures
13. Only Talk to Your Friends
14. Refactor
15. Don't Overengineer
16. Don't Overuse Indentation
17. Small is Beautiful
18. Use Metrics
19. Boy Scout Rule: Leave Camp Cleaner Than You Found It

Unix Philosophy

1. Simple's Better Than Complex
2. **Small is Beautiful (*Again*)**
3. Make Each Program Do One Thing Well
4. Build a Prototype First
5. Portability Over Efficiency
6. Store Data in Flat Text Files
7. Use Software Leverage
8. Avoid Captive User Interfaces
9. **Program = Filter**
10. Worse is Better
11. Clean > Clever Code
12. **Design *Connected* Programs**
13. Make Your Code Robust
14. Repair What You Can — But Fail Early and Noisily
15. Write Programs to Write Programs

Premature Optimization

"Programmers waste enormous amounts of time thinking about […] the speed of noncritical parts of their programs. We should forget about small efficiencies, say about 97 % of the time: premature optimization is the root of all evil." – **Donald Knuth**

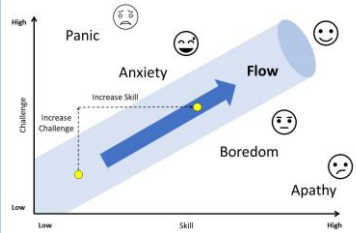
Performance Tuning 101
1. Measure, then improve
2. Focus on the slow 20%
3. Algorithmic optimization wins
4. All hail to the cache
5. Solve an easier problem version
6. Know when to stop

Less Is More in Design

How to Simplify Design?
1. Use whitespace
2. Remove design elements
3. Remove features
4. Reduce variation of fonts, font types, colors
5. Be consistent across UIs

Flow

"… the source code of ultimate human performance" – ***Kotler***

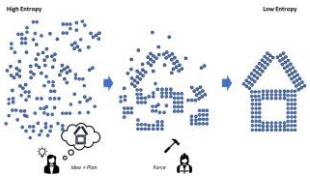


How to Achieve Flow? (1) clear goals, (2) immediate feedback, and (3) balance opportunity & capacity.

Flow Tips for Coders
1. Always work on an explicit practical code project
2. Work on fun projects that fulfill your purpose
3. Perform from your strengths
4. Big chunks of coding time
5. Reduce distractions: smartphone + social
6. Sleep a lot, eat healthily, read quality books, and exercise → garbage in, garbage out!

Focus

You can take raw resources and move them from a state of high entropy into a state of low entropy—using *focused effort towards the attainment of a greater plan*.



3-Step Approach of Efficient Software Creation
1. Plan your code
2. Apply focused effort to make it real.
3. Seek feedback

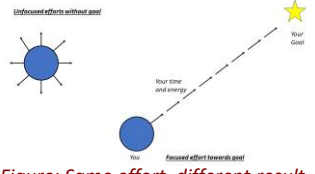


Figure: Same effort, different result.

*Subscribe to the **11x FREE** Python Cheat Sheet Course:*
https://blog.finxter.com/python-cheat-sheets/

Data Science Cheat Sheet

Python Regular Expressions

SPECIAL CHARACTERS

`^` | Matches the expression to its right at the start of a string. It matches every such instance before each `\n` in the string.

`$` | Matches the expression to its left at the end of a string. It matches every such instance before each `\n` in the string.

`.` | Matches any character except line terminators like `\n`.

`\` | Escapes special characters or denotes character classes.

`A|B` | Matches expression `A` or `B`. If A is matched first, B is left untried.

`+` | Greedily matches the expression to its left 1 or more times.

`*` | Greedily matches the expression to its left 0 or more times.

`?` | Greedily matches the expression to its left 0 or 1 times. But if `?` is added to qualifiers (`+`, `*`, and `?` itself) it will perform matches in a non-greedy manner.

`{m}` | Matches the expression to its left `m` times, and not less.

`{m,n}` | Matches the expression to its left `m` to `n` times, and not less.

`{m,n}?` | Matches the expression to its left `m` times, and ignores `n`. See `?` above.

CHARACTER CLASSES [A.K.A. SPECIAL SEQUENCES]

`\w` | Matches alphanumeric characters, which means `a-z`, `A-Z`, and `0-9`. It also matches the underscore, `_`.

`\d` | Matches digits, which means `0-9`.

`\D` | Matches any non-digits.

`\s` | Matches whitespace characters, which include the `\t`, `\n`, `\r`, and space characters.

`\S` | Matches non-whitespace characters.

`\b` | Matches the boundary (or empty string) at the start and end of a word, that is, between `\w` and `\W`.

`\B` | Matches where `\b` does not, that is, the boundary of `\w` characters.

`\A` | Matches the expression to its right at the absolute start of a string whether in single or multi-line mode.

`\Z` | Matches the expression to its left at the absolute end of a string whether in single or multi-line mode.

SETS

`[ ]` | Contains a set of characters to match.

`[amk]` | Matches either `a`, `m`, or `k`. It does not match `amk`.

`[a-z]` | Matches any alphabet from `a` to `z`.

`[a\-z]` | Matches `a`, `-`, or `z`. It matches `-` because `\` escapes it.

`[a-]` | Matches `a` or `-`, because `-` is not being used to indicate a series of characters.

`[-a]` | As above, matches `a` or `-`.

`[a-z0-9]` | Matches characters from `a` to `z` and also from `0` to `9`.

`[(+*)]` | Special characters become literal inside a set, so this matches `(`, `+`, `*`, and `)`.

`[^ab5]` | Adding `^` excludes any character in the set. Here, it matches characters that are not `a`, `b`, or `5`.

GROUPS

`( )` | Matches the expression inside the parentheses and groups it.

`(?)` | Inside parentheses like this, `?` acts as an extension notation. Its meaning depends on the character immediately to its right.

`(?PAB)` | Matches the expression `AB`, and it can be accessed with the group name.

`(?aiLmsux)` | Here, `a`, `i`, `L`, `m`, `s`, `u`, and `x` are flags:

- `a` — Matches ASCII only
- `i` — Ignore case
- `L` — Locale dependent
- `m` — Multi-line
- `s` — Matches all
- `u` — Matches unicode
- `x` — Verbose

`(?:A)` | Matches the expression as represented by `A`, but unlike `(?PAB)`, it cannot be retrieved afterwards.

`(?#...)` | A comment. Contents are for us to read, not for matching.

`A(?=B)` | Lookahead assertion. This matches the expression `A` only if it is followed by `B`.

`A(?!B)` | Negative lookahead assertion. This matches the expression `A` only if it is not followed by `B`.

`(?<=B)A` | Positive lookbehind assertion. This matches the expression `A` only if `B` is immediately to its left. This can only matched fixed length expressions.

`(?<!B)A` | Negative lookbehind assertion. This matches the expression `A` only if `B` is not immediately to its left. This can only matched fixed length expressions.

`(?P=name)` | Matches the expression matched by an earlier group named "name".

`(...)\1` | The number `1` corresponds to the first group to be matched. If we want to match more instances of the same expression, simply use its number instead of writing out the whole expression again. We can use from `1` up to `99` such groups and their corresponding numbers.

POPULAR PYTHON RE MODULE FUNCTIONS

`re.findall(A, B)` | Matches all instances of an expression `A` in a string `B` and returns them in a list.

`re.search(A, B)` | Matches the first instance of an expression `A` in a string `B`, and returns it as a re match object.

`re.split(A, B)` | Split a string `B` into a list using the delimiter `A`.

`re.sub(A, B, C)` | Replace `A` with `B` in the string `C`.

Data Science Cheat Sheet

Python Basics

BASICS, PRINTING AND GETTING HELP

`x = 3` - Assign **3** to the variable **x**
`print(x)` - Print the value of **x**
`type(x)` - Return the type of the variable **x** (in this case, **int** for integer)
`help(x)` - Show documentation for the **str** data type
`help(print)` - Show documentation for the **print()** function

READING FILES

```
f = open("my_file.txt","r")
file_as_string = f.read()
```

- Open the file **my_file.txt** and assign its contents to **s**

```
import csv
f = open("my_dataset.csv","r")
csvreader = csv.reader(f)
csv_as_list = list(csvreader)
```

- Open the CSV file **my_dataset.csv** and assign its data to the list of lists **csv_as_list**

STRINGS

`s = "hello"` - Assign the string **"hello"** to the variable **s**

```
s = """She said,
"there's a good idea."
"""
```

- Assign a multi-line string to the variable **s**. Also used to create strings that contain both **"** and **'** characters

`len(s)` - Return the number of characters in **s**
`s.startswith("hel")` - Test whether **s** starts with the substring **"hel"**
`s.endswith("lo")` - Test whether **s** ends with the substring **"lo"**
`"{} plus {} is {}".format(3,1,4)` - Return the string with the values **3**, **1**, and **4** inserted
`s.replace("e","z")` - Return a new string based on **s** with all occurances of **"e"** replaced with **"z"**
`s.split(" ")` - Split the string **s** into a list of strings, separating on the character **" "** and return that list

NUMERIC TYPES AND MATHEMATICAL OPERATIONS

`i = int("5")` - Convert the string **"5"** to the integer **5** and assign the result to **i**
`f = float("2.5")` - Convert the string **"2.5"** to the float value **2.5** and assign the result to **f**
`5 + 5` - Addition
`5 - 5` - Subtraction
`10 / 2` - Division
`5 * 2` - Multiplication
`3 ** 2` - Raise **3** to the power of **2** (or 3^2)
`27 ** (1/3)` - The **3**rd root of **27** (or $\sqrt[3]{27}$)
`x += 1` - Assign the value of **x + 1** to **x**
`x -= 1` - Assign the value of **x - 1** to **x**

LISTS

`l = [100,21,88,3]` - Assign a list containing the integers **100**, **21**, **88**, and **3** to the variable **l**
`l = list()` - Create an empty list and assign the result to **l**
`l[0]` - Return the first value in the list **l**
`l[-1]` - Return the last value in the list **l**
`l[1:3]` - Return a slice (list) containing the second and third values of **l**
`len(l)` - Return the number of elements in **l**
`sum(l)` - Return the sum of the values of **l**
`min(l)` - Return the minimum value from **l**
`max(l)` - Return the maximum value from **l**
`l.append(16)` - Append the value **16** to the end of **l**
`l.sort()` - Sort the items in **l** in ascending order
`" ".join(["A","B","C","D"])` - Converts the list **["A", "B", "C", "D"]** into the string **"A B C D"**

DICTIONARIES

`d = {"CA":"Canada","GB":"Great Britain", "IN":"India"}` - Create a dictionary with keys of **"CA"**, **"GB"**, and **"IN"** and corresponding values of of **"Canada"**, **"Great Britain"**, and **"India"**
`d["GB"]` - Return the value from the dictionary **d** that has the key **"GB"**
`d.get("AU","Sorry")` - Return the value from the dictionary **d** that has the key **"AU"**, or the string **"Sorry"** if the key **"AU"** is not found in **d**
`d.keys()` - Return a list of the keys from **d**
`d.values()` - Return a list of the values from **d**
`d.items()` - Return a list of **(key, value)** pairs from **d**

MODULES AND FUNCTIONS

The body of a function is defined through indentation.

`import random` - Import the module **random**
`from math import sqrt` - Import the function **sqrt** from the module **math**

```
def calculate(addition_one,addition_two,
exponent=1,factor=1):
  result = (value_one + value_two) ** exponent * factor
  return result
```

- Define a new function **calculate** with two required and two optional named arguments which calculates and returns a result.

`addition(3,5,factor=10)` - Run the **addition** function with the values **3** and **5** and the named argument **10**

BOOLEAN COMPARISONS

`x == 5` - Test whether **x** is equal to **5**
`x != 5` - Test whether **x** is not equal to **5**
`x > 5` - Test whether **x** is greater than **5**
`x < 5` - Test whether **x** is less than **5**
`x >= 5` - Test whether **x** is greater than or equal to **5**
`x <= 5` - Test whether **x** is less than or equal to **5**
`x == 5 or name == "alfred"` - Test whether **x** is equal to **5** or **name** is equal to **"alfred"**
`x == 5 and name == "alfred"` - Test whether **x** is equal to **5** and **name** is equal to **"alfred"**
`5 in l` - Checks whether the value **5** exists in the list **l**
`"GB" in d` - Checks whether the value **"GB"** exists in the keys for **d**

IF STATEMENTS AND LOOPS

The body of if statements and loops are defined through indentation.

```
if x > 5:
  print("{} is greater than five".format(x))
elif x < 0:
  print("{} is negative".format(x))
else:
  print("{} is between zero and five".format(x))
```

- Test the value of the variable **x** and run the code body based on the value

```
for value in l:
  print(value)
```

- Iterate over each value in **l**, running the code in the body of the loop with each iteration

```
while x < 10:
  x += 1
```

- Run the code in the body of the loop until the value of **x** is no longer less than **10**

Data Science Cheat Sheet

Python - Intermediate

KEY BASICS, PRINTING AND GETTING HELP

This cheat sheet assumes you are familiar with the content of our Python Basics Cheat Sheet

`s` - A Python string variable
`i` - A Python integer variable
`f` - A Python float variable
`l` - A Python list variable
`d` - A Python dictionary variable

LISTS

`l.pop(3)` - Returns the fourth item from `l` and deletes it from the list
`l.remove(x)` - Removes the first item in `l` that is equal to `x`
`l.reverse()` - Reverses the order of the items in `l`
`l[1::2]` - Returns every second item from `l`, commencing from the `1`st item
`l[-5:]` - Returns the last 5 items from `l` specific axis

STRINGS

`s.lower()` - Returns a lowercase version of `s`
`s.title()` - Returns `s` with the first letter of every word capitalized
`"23".zfill(4)` - Returns `"0023"` by left-filling the string with `0`'s to make it's length `4`.
`s.splitlines()` - Returns a list by splitting the string on any newline characters.

Python strings share some common methods with lists

`s[:5]` - Returns the first `5` characters of `s`
`"fri" + "end"` - Returns `"friend"`
`"end" in s` - Returns `True` if the substring `"end"` is found in `s`

RANGE

Range objects are useful for creating sequences of integers for looping.

`range(5)` - Returns a sequence from `0` to `4`
`range(2000,2018)` - Returns a sequence from `2000` to `2017`
`range(0,11,2)` - Returns a sequence from `0` to `10`, with each item incrementing by `2`
`range(0,-10,-1)` - Returns a sequence from `0` to `-9`
`list(range(5))` - Returns a list from `0` to `4`

DICTIONARIES

`max(d, key=d.get)` - Return the key that corresponds to the largest value in `d`
`min(d, key=d.get)` - Return the key that corresponds to the smallest value in `d`

SETS

`my_set = set(l)` - Return a `set` object containing the unique values from `l`
`len(my_set)` - Returns the number of objects in `my_set` (or, the number of unique values from `l`)
`a in my_set` - Returns `True` if the value `a` exists in `my_set`

REGULAR EXPRESSIONS

`import re` - Import the Regular Expressions module
`re.search("abc",s)` - Returns a `match` object if the regex `"abc"` is found in `s`, otherwise `None`
`re.sub("abc","xyz",s)` - Returns a string where all instances matching regex `"abc"` are replaced by `"xyz"`

LIST COMPREHENSION

A one-line expression of a for loop

`[i ** 2 for i in range(10)]` - Returns a list of the squares of values from `0` to `9`
`[s.lower() for s in l_strings]` - Returns the list `l_strings`, with each item having had the `.lower()` method applied
`[i for i in l_floats if i < 0.5]` - Returns the items from `l_floats` that are less than `0.5`

FUNCTIONS FOR LOOPING

```
for i, value in enumerate(l):
  print("The value of item {} is {}".
  format(i,value))
```

- Iterate over the list `l`, printing the index location of each item and its value

```
for one, two in zip(l_one,l_two):
  print("one: {}, two: {}".format(one,two))
```

- Iterate over two lists, `l_one` and `l_two` and print each value

```
while x < 10:
  x += 1
```

- Run the code in the body of the loop until the value of `x` is no longer less than `10`

DATETIME

`import datetime as dt` - Import the `datetime` module
`now = dt.datetime.now()` - Assign `datetime` object representing the current time to `now`
`wks4 = dt.datetime.timedelta(weeks=4)` - Assign a `timedelta` object representing a timespan of 4 weeks to `wks4`
`now - wks4` - Return a `datetime` object representing the time 4 weeks prior to `now`
`newyear_2020 = dt.datetime(year=2020, month=12, day=31)` - Assign a `datetime` object representing December 25, 2020 to `newyear_2020`
`newyear_2020.strftime("%A, %b %d, %Y")` - Returns `"Thursday, Dec 31, 2020"`
`dt.datetime.strptime('Dec 31, 2020',"%b %d, %Y")` - Return a `datetime` object representing December 31, 2020

RANDOM

`import random` - Import the `random` module
`random.random()` - Returns a random float between `0.0` and `1.0`
`random.randint(0,10)` - Returns a random integer between `0` and `10`
`random.choice(l)` - Returns a random item from the list `l`

COUNTER

`from collections import Counter` - Import the `Counter` class
`c = Counter(l)` - Assign a `Counter` (dict-like) object with the counts of each unique item from `l`, to `c`
`c.most_common(3)` - Return the 3 most common items from `l`

TRY/EXCEPT

Catch and deal with Errors

`l_ints = [1, 2, 3, "", 5]` - Assign a list of integers with one missing value to `l_ints`

```
l_floats = []
for i in l_ints:
  try:
    l_floats.append(float(i))
  except:
    l_floats.append(i)
```

- Convert each value of `l_ints` to a float, catching and handling `ValueError: could not convert string to float:` where values are missing.

Data Science Cheat Sheet

NumPy

KEY

We'll use shorthand in this cheat sheet

`arr` - A numpy Array object

IMPORTS

Import these to start

```
import numpy as np
```

IMPORTING/EXPORTING

`np.loadtxt('file.txt')` - From a text file

`np.genfromtxt('file.csv',delimiter=',')` - From a CSV file

`np.savetxt('file.txt',arr,delimiter=' ')` - Writes to a text file

`np.savetxt('file.csv',arr,delimiter=',')` - Writes to a CSV file

CREATING ARRAYS

`np.array([1,2,3])` - One dimensional array

`np.array([(1,2,3),(4,5,6)])` - Two dimensional array

`np.zeros(3)` - 1D array of length **3** all values **0**

`np.ones((3,4))` - **3**x**4** array with all values **1**

`np.eye(5)` - **5**x**5** array of **0** with **1** on diagonal (Identity matrix)

`np.linspace(0,100,6)` - Array of **6** evenly divided values from **0** to **100**

`np.arange(0,10,3)` - Array of values from **0** to less than **10** with step **3** (eg **[0,3,6,9]**)

`np.full((2,3),8)` - **2**x**3** array with all values **8**

`np.random.rand(4,5)` - **4**x**5** array of random floats between **0**-**1**

`np.random.rand(6,7)*100` - **6**x**7** array of random floats between **0**-**100**

`np.random.randint(5,size=(2,3))` - **2**x**3** array with random ints between **0**-**4**

INSPECTING PROPERTIES

`arr.size` - Returns number of elements in `arr`

`arr.shape` - Returns dimensions of `arr` (rows, columns)

`arr.dtype` - Returns type of elements in `arr`

`arr.astype(dtype)` - Convert `arr` elements to type `dtype`

`arr.tolist()` - Convert `arr` to a Python list

`np.info(np.eye)` - View documentation for `np.eye`

COPYING/SORTING/RESHAPING

`np.copy(arr)` - Copies `arr` to new memory

`arr.view(dtype)` - Creates view of `arr` elements with type `dtype`

`arr.sort()` - Sorts `arr`

`arr.sort(axis=0)` - Sorts specific axis of `arr`

`two_d_arr.flatten()` - Flattens 2D array `two_d_arr` to 1D

`arr.T` - Transposes `arr` (rows become columns and vice versa)

`arr.reshape(3,4)` - Reshapes `arr` to **3** rows, **4** columns without changing data

`arr.resize((5,6))` - Changes `arr` shape to **5**x**6** and fills new values with **0**

ADDING/REMOVING ELEMENTS

`np.append(arr,values)` - Appends `values` to end of `arr`

`np.insert(arr,2,values)` - Inserts `values` into `arr` before index **2**

`np.delete(arr,3,axis=0)` - Deletes row on index **3** of `arr`

`np.delete(arr,4,axis=1)` - Deletes column on index **4** of `arr`

COMBINING/SPLITTING

`np.concatenate((arr1,arr2),axis=0)` - Adds `arr2` as rows to the end of `arr1`

`np.concatenate((arr1,arr2),axis=1)` - Adds `arr2` as columns to end of `arr1`

`np.split(arr,3)` - Splits `arr` into **3** sub-arrays

`np.hsplit(arr,5)` - Splits `arr` horizontally on the **5**th index

INDEXING/SLICING/SUBSETTING

`arr[5]` - Returns the element at index **5**

`arr[2,5]` - Returns the 2D array element on index **[2][5]**

`arr[1]=4` - Assigns array element on index **1** the value **4**

`arr[1,3]=10` - Assigns array element on index **[1][3]** the value **10**

`arr[0:3]` - Returns the elements at indices **0,1,2** (On a 2D array: returns rows **0,1,2**)

`arr[0:3,4]` - Returns the elements on rows **0,1,2** at column **4**

`arr[:2]` - Returns the elements at indices **0,1** (On a 2D array: returns rows **0,1**)

`arr[:,1]` - Returns the elements at index **1** on all rows

`arr<5` - Returns an array with boolean values

`(arr1<3) & (arr2>5)` - Returns an array with boolean values

`~arr` - Inverts a boolean array

`arr[arr<5]` - Returns array elements smaller than **5**

SCALAR MATH

`np.add(arr,1)` - Add **1** to each array element

`np.subtract(arr,2)` - Subtract **2** from each array element

`np.multiply(arr,3)` - Multiply each array element by **3**

`np.divide(arr,4)` - Divide each array element by **4** (returns `np.nan` for division by zero)

`np.power(arr,5)` - Raise each array element to the **5**th power

VECTOR MATH

`np.add(arr1,arr2)` - Elementwise add `arr2` to `arr1`

`np.subtract(arr1,arr2)` - Elementwise subtract `arr2` from `arr1`

`np.multiply(arr1,arr2)` - Elementwise multiply `arr1` by `arr2`

`np.divide(arr1,arr2)` - Elementwise divide `arr1` by `arr2`

`np.power(arr1,arr2)` - Elementwise raise `arr1` raised to the power of `arr2`

`np.array_equal(arr1,arr2)` - Returns `True` if the arrays have the same elements and shape

`np.sqrt(arr)` - Square root of each element in the array

`np.sin(arr)` - Sine of each element in the array

`np.log(arr)` - Natural log of each element in the array

`np.abs(arr)` - Absolute value of each element in the array

`np.ceil(arr)` - Rounds up to the nearest int

`np.floor(arr)` - Rounds down to the nearest int

`np.round(arr)` - Rounds to the nearest int

STATISTICS

`np.mean(arr,axis=0)` - Returns mean along specific axis

`arr.sum()` - Returns sum of `arr`

`arr.min()` - Returns minimum value of `arr`

`arr.max(axis=0)` - Returns maximum value of specific axis

`np.var(arr)` - Returns the variance of array

`np.std(arr,axis=1)` - Returns the standard deviation of specific axis

`arr.corrcoef()` - Returns correlation coefficient of array

Data Science Cheat Sheet

Pandas

KEY

We'll use shorthand in this cheat sheet

`df` - A pandas DataFrame object

`s` - A pandas Series object

IMPORTS

Import these to start

```
import pandas as pd
import numpy as np
```

IMPORTING DATA

`pd.read_csv(filename)` - From a CSV file

`pd.read_table(filename)` - From a delimited text file (like TSV)

`pd.read_excel(filename)` - From an Excel file

`pd.read_sql(query, connection_object)` - Reads from a SQL table/database

`pd.read_json(json_string)` - Reads from a JSON formatted string, URL or file.

`pd.read_html(url)` - Parses an html URL, string or file and extracts tables to a list of dataframes

`pd.read_clipboard()` - Takes the contents of your clipboard and passes it to `read_table()`

`pd.DataFrame(dict)` - From a dict, keys for columns names, values for data as lists

EXPORTING DATA

`df.to_csv(filename)` - Writes to a CSV file

`df.to_excel(filename)` - Writes to an Excel file

`df.to_sql(table_name, connection_object)` - Writes to a SQL table

`df.to_json(filename)` - Writes to a file in JSON format

`df.to_html(filename)` - Saves as an HTML table

`df.to_clipboard()` - Writes to the clipboard

CREATE TEST OBJECTS

Useful for testing

`pd.DataFrame(np.random.rand(20,5))` - 5 columns and 20 rows of random floats

`pd.Series(my_list)` - Creates a series from an iterable `my_list`

`df.index = pd.date_range('1900/1/30', periods=df.shape[0])` - Adds a date index

VIEWING/INSPECTING DATA

`df.head(n)` - First `n` rows of the DataFrame

`df.tail(n)` - Last `n` rows of the DataFrame

`df.shape` - Number of rows and columns

`df.info()` - Index, Datatype and Memory information

`df.describe()` - Summary statistics for numerical columns

`s.value_counts(dropna=False)` - Views unique values and counts

`df.apply(pd.Series.value_counts)` - Unique values and counts for all columns

SELECTION

`df[col]` - Returns column with label `col` as Series

`df[[col1, col2]]` - Returns Columns as a new DataFrame

`s.iloc[0]` - Selection by position

`s.loc[0]` - Selection by index

`df.iloc[0,:]` - First row

`df.iloc[0,0]` - First element of first column

DATA CLEANING

`df.columns = ['a','b','c']` - Renames columns

`pd.isnull()` - Checks for null Values, Returns Boolean Array

`pd.notnull()` - Opposite of `s.isnull()`

`df.dropna()` - Drops all rows that contain null values

`df.dropna(axis=1)` - Drops all columns that contain null values

`df.dropna(axis=1,thresh=n)` - Drops all rows have have less than `n` non null values

`df.fillna(x)` - Replaces all null values with `x`

`s.fillna(s.mean())` - Replaces all null values with the mean (mean can be replaced with almost any function from the statistics section)

`s.astype(float)` - Converts the datatype of the series to float

`s.replace(1,'one')` - Replaces all values equal to `1` with `'one'`

`s.replace([1,3],['one','three'])` - Replaces all `1` with `'one'` and `3` with `'three'`

`df.rename(columns=lambda x: x + 1)` - Mass renaming of columns

`df.rename(columns={'old_name': 'new_name'})` - Selective renaming

`df.set_index('column_one')` - Changes the index

`df.rename(index=lambda x: x + 1)` - Mass renaming of index

FILTER, SORT, & GROUPBY

`df[df[col] > 0.5]` - Rows where the `col` column is greater than `0.5`

`df[(df[col] > 0.5) & (df[col] < 0.7)]` - Rows where `0.7 > col > 0.5`

`df.sort_values(col1)` - Sorts values by `col1` in ascending order

`df.sort_values(col2,ascending=False)` - Sorts values by `col2` in descending order

`df.sort_values([col1,col2], ascending=[True,False])` - Sorts values by `col1` in ascending order then `col2` in descending order

`df.groupby(col)` - Returns a groupby object for values from one column

`df.groupby([col1,col2])` - Returns a groupby object values from multiple columns

`df.groupby(col1)[col2].mean()` - Returns the mean of the values in `col2`, grouped by the values in `col1` (mean can be replaced with almost any function from the statistics section)

`df.pivot_table(index=col1,values=[col2,col3],aggfunc=mean)` - Creates a pivot table that groups by `col1` and calculates the mean of `col2` and `col3`

`df.groupby(col1).agg(np.mean)` - Finds the average across all columns for every unique column 1 group

`df.apply(np.mean)` - Applies a function across each column

`df.apply(np.max, axis=1)` - Applies a function across each row

JOIN/COMBINE

`df1.append(df2)` - Adds the rows in `df1` to the end of `df2` (columns should be identical)

`pd.concat([df1, df2],axis=1)` - Adds the columns in `df1` to the end of `df2` (rows should be identical)

`df1.join(df2,on=col1,how='inner')` - SQL-style joins the columns in `df1` with the columns on `df2` where the rows for `col` have identical values. `how` can be one of `'left'`, `'right'`, `'outer'`, `'inner'`

STATISTICS

These can all be applied to a series as well.

`df.describe()` - Summary statistics for numerical columns

`df.mean()` - Returns the mean of all columns

`df.corr()` - Returns the correlation between columns in a DataFrame

`df.count()` - Returns the number of non-null values in each DataFrame column

`df.max()` - Returns the highest value in each column

`df.min()` - Returns the lowest value in each column

`df.median()` - Returns the median of each column

`df.std()` - Returns the standard deviation of each column

datacamp

Python For Data Science

python™ Basics Cheat Sheet

Learn Python Basics online at www.DataCamp.com

> Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

```
>>> x+2 #Sum of two variables
 7
>>> x-2 #Subtraction of two variables
 3
>>> x*2 #Multiplication of two variables
 10
>>> x**2 #Exponentiation of a variable
 25
>>> x%2 #Remainder of a variable
 1
>>> x/float(2) #Division of a variable
 2.5
```

Types and Type Conversion

str()

```
'5', '3.45', 'True' #Variables to strings
```

int()

```
5, 3, 1 #Variables to integers
```

float()

```
5.0, 1.0 #Variables to floats
```

bool()

```
True, True, True #Variables to booleans
```

> Libraries

pandas — Data analysis

NumPy — Scientific computing

matplotlib — 2D plotting

learn — Machine learning

Import Libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

> Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
 'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Indexing

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper() #String to uppercase
>>> my_string.lower() #String to lowercase
>>> my_string.count('w') #Count String elements
>>> my_string.replace('e', 'i') #Replace String elements
>>> my_string.strip() #Strip whitespaces
```

> NumPy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset

```
>>> my_array[1] #Select item at index 1
 2
```

Slice

```
>>> my_array[0:2] #Select items at index 0 and 1
 array([1, 2])
```

Subset 2D Numpy arrays

```
>>> my_2darray[:,0] #my_2darray[rows, columns]
 array([1, 4])
```

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False, True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape #Get the dimensions of the array
>>> np.append(other_array) #Append items to an array
>>> np.insert(my_array, 1, 5) #Insert items in an array
>>> np.delete(my_array,[1]) #Delete items in an array
>>> np.mean(my_array) #Mean of the array
>>> np.median(my_array) #Median of the array
>>> my_array.corrcoef() #Correlation coefficient
>>> np.std(my_array) #Standard deviation
```

> Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1] #Select item at index 1
>>> my_list[-3] #Select 3rd last item
```

Slice

```
>>> my_list[1:3] #Select items at index 1 and 2
>>> my_list[1:] #Select items after index 0
>>> my_list[:3] #Select items before index 3
>>> my_list[:] #Copy my_list
```

Subset Lists of Lists

```
>>> my_list2[1][0] #my_list[list][itemOfList]
>>> my_list2[1][:2]
```

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a) #Get the index of an item
>>> my_list.count(a) #Count an item
>>> my_list.append('!') #Append an item at a time
>>> my_list.remove('!') #Remove an item
>>> del(my_list[0:1]) #Remove an item
>>> my_list.reverse() #Reverse the list
>>> my_list.extend('!') #Append an item
>>> my_list.pop(-1) #Remove an item
>>> my_list.insert(0,'!') #Insert an item
>>> my_list.sort() #Sort the list
```

> Python IDEs (Integrated Development Environment)

ANACONDA. — Leading open data science platform powered by Python

SPYDER — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code

> Asking For Help

```
>>> help(str)
```

datacamp

Python For Data Science
Importing Data Cheat Sheet

Learn Python online at www.DataCamp.com

> Importing Data in Python

Most of the time, you'll use either NumPy or pandas to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

> Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

> Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Using the context manager with

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline()) #Read a single line
        print(file.readline())
        print(file.readline())
```

Table Data: Flat Files

Importing Flat Files with NumPy

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',', #String used to separate values
                      skiprows=2, #Skip the first 2 lines
                      usecols=[0,2], #Read the 1st and 3rd column
                      dtype=str) #The type of the resulting array
```

Files with mixed data type

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True, #Look for column header
                         dtype=None)
>>> data_array = np.recfromcsv(filename)
#The default dtype of the np.recfromcsv() function is None
```

Importing Flat Files with Pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5, #Number of rows of file to read
                       header=None, #Row number to use as col names
                       sep='\t', #Delimiter to use
                       comment='#', #Character to split comments
                       na_values=[""]) #String to recognize as NA/NaN
```

> Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype #Data type of array elements
>>> data_array.shape #Array dimensions
>>> len(data_array) #Length of array
```

Pandas DataFrames

```
>>> df.head() #Return first DataFrame rows
>>> df.tail() #Return last DataFrame rows
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> data_array = data.values #Convert a DataFrame to an a NumPy array
```

> SAS File

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

> Stata File

```
>>> data = pd.read_stata('urbanpop.dta')
```

> Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                           'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` **attribute:**

```
>>> data.sheet_names
```

> Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

Use the `table_names()` **method to fetch a list of table names:**

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager with

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

> Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

> Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

> HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

> Exploring Dictionaries

Querying relational databases with pandas

```
>>> print(mat.keys()) #Print dictionary keys
>>> for key in data.keys(): #Print dictionary keys
        print(key)
meta
quality
strain
>>> pickled_data.values() #Return dictionary values
>>> print(mat.items()) #Returns items in list format of (key, value) tuple pairs
```

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys() #Explore the HDF5
structure
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
#Retrieve the value for a key
>>> print(data['meta']['Description'].value)
```

> Navigating Your FileSystem

Magic Commands

```
!ls  #List directory contents of files and directories
%cd .. #Change current working directory
%pwd #Return the current working directory path
```

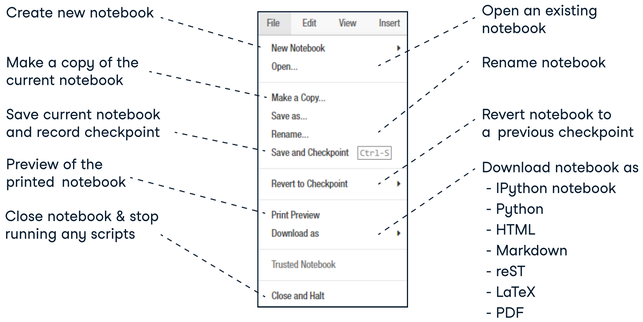
OS Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd() #Store the name of current directory in a string
>>> os.listdir(wd) #Output contents of the directory in a list
>>> os.chdir(path) #Change current working directory
>>> os.rename("test1.txt", #Rename a file
              "test2.txt")
>>> os.remove("test1.txt") #Delete an existing file
>>> os.mkdir("newdir") #Create a new directory
```

Python For Data Science
Jupyter Cheat Sheet

Learn Jupyter online at www.DataCamp.com

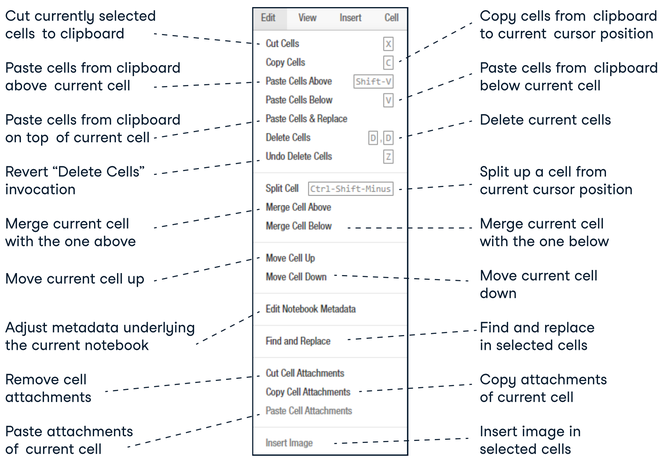
> Saving/Loading Notebooks



> Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells

Edit Cells



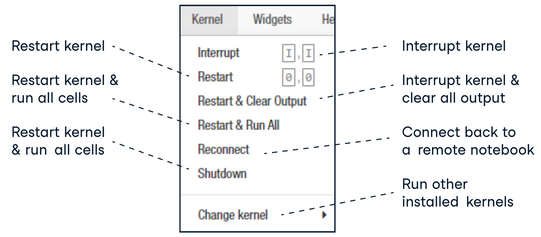
Insert Cells

> Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython IRkernel IJulia

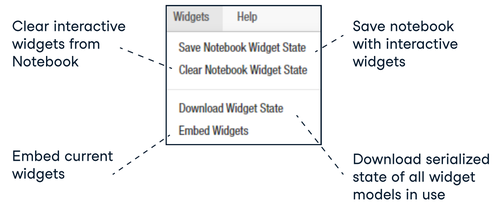
Installing Jupyter Notebook will automatically install the IPython kernel.



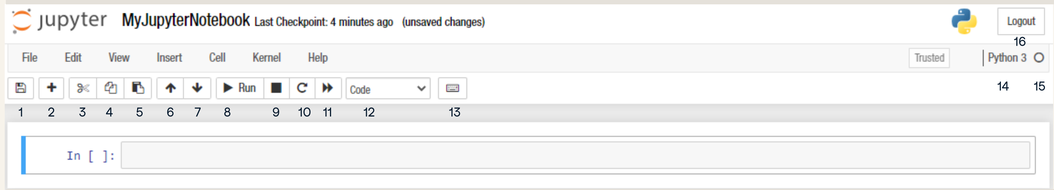
> Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

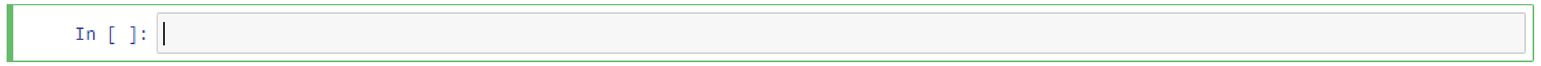


Command Mode:

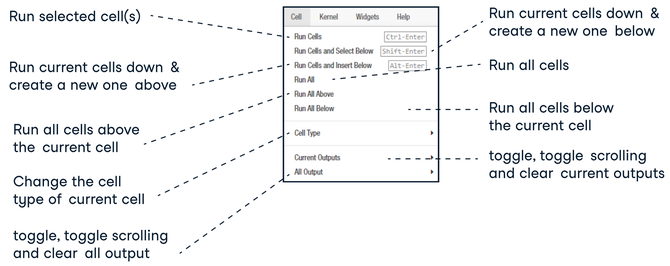


1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Restart kernel and re-run notebook
12. Display characteristics
13. Open command palette
14. Current kernel
15. Kernel status
16. Log out from notebook server

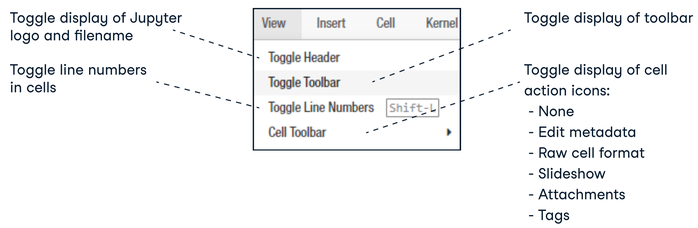
Edit Mode:



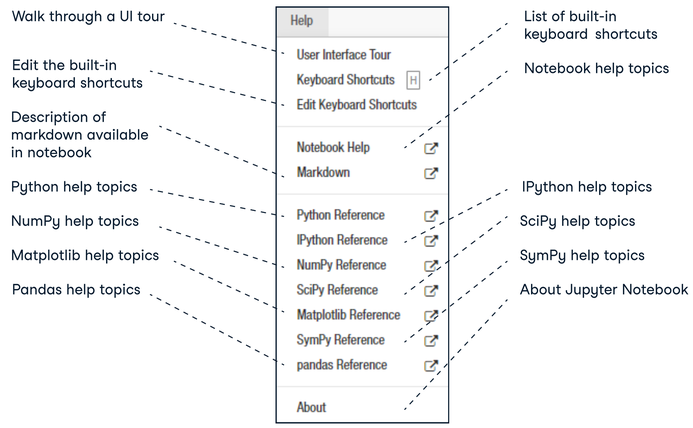
Executing Cells



View Cells



Asking For Help



datacamp

Python For Data Science

Pandas Basics Cheat Sheet

Learn Pandas Basics online at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

Use the following import convention:

```
>>> import pandas as pd
```

> Pandas Data Structures

Series

A **one-dimensional** labeled array capable of holding any data type

Index →

a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Dataframe

A **two-dimensional** labeled data structure with columns of potentially different types

Columns →

Index →	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

> Dropping

```
>>> s.drop(['a', 'c']) #Drop values from rows (axis=0)
>>> df.drop('Country', axis=1) #Drop values from columns(axis=1)
```

> Asking For Help

```
>>> help(pd.Series.loc)
```

> Sort & Rank

```
>>> df.sort_index() #Sort by labels along an axis
>>> df.sort_values(by='Country') #Sort by the values along an axis
>>> df.rank() #Assign ranks to entries
```

> I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql() **is a convenience wrapper around** read_sql_table() **and** read_sql_query()

```
>>> df.to_sql('myDf', engine)
```

> Selection

Also see NumPy Arrays

Getting

```
>>> s['b'] #Get one element
 -5
>>> df[1:] #Get subset of a DataFrame
   Country Capital Population
 1 India New Delhi 1303171035
 2 Brazil Brasília 207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
 Country Brazil
 Capital Brasília
 Population 207847528
>>> df.ix[:,'Capital'] #Select a single column of subset of columns
 0 Brussels
 1 New Delhi
 2 Brasília
>>> df.ix[1,'Capital'] #Select rows and columns
 'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>1200000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```

> Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape #(rows,columns)
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> df.count() #Number of non-NA values
```

Summary

```
>>> df.sum() #Sum of values
>>> df.cumsum() #Cummulative sum of values
>>> df.min()/df.max() #Minimum/maximum values
>>> df.idxmin()/df.idxmax() #Minimum/Maximum index value
>>> df.describe() #Summary statistics
>>> df.mean() #Mean of values
>>> df.median() #Median of values
```

> Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f) #Apply function
>>> df.applymap(f) #Apply function element-wise
```

> Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
 a 10.0
 b NaN
 c 5.0
 d 7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_values=0)
 a 10.0
 b -5.0
 c 5.0
 d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

datacamp

Python For Data Science

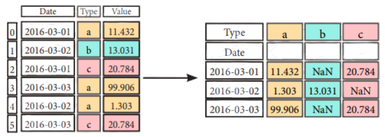
Data Wrangling in Pandas Cheat Sheet

Learn Data Wrangling online at www.DataCamp.com

> Reshaping Data

Pivot

```
>>> df3= df2.pivot(index='Date', #Spread rows into columns
                   columns='Type',
                   values='Value')
```

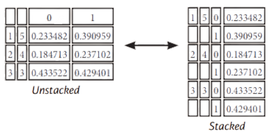


Pivot Table

```
>>> df4 = pd.pivot_table(df2, #Spread rows into
                         columns values='Value',
                         index='Date',
                         columns='Type'])
```

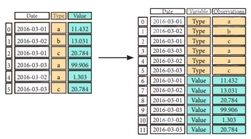
Stack / Unstack

```
>>> stacked = df5.stack() #Pivot a level of column labels
>>> stacked.unstack() #Pivot a level of index labels
```



Melt

```
>>> pd.melt(df2, #Gather columns into rows
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```



> Iteration

```
>>> df.iteritems() #(Column-index, Series) pairs
>>> df.iterrows() #(Row-index, Series) pairs
```

> Missing Data

```
>>> df.dropna() #Drop NaN values
>>> df3.fillna(df3.mean()) #Fill NaN values with a predetermined value
>>> df2.replace("a", "f") #Replace values with others
```

> Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()] #Select cols with any vals >1
>>> df3.loc[:,(df3>1).all()] #Select cols with vals > 1
>>> df3.loc[:,df3.isnull().any()] #Select cols with NaN
>>> df3.loc[:,df3.notnull().all()] #Select cols without NaN
```

Indexing With `isin()`

```
>>> df[(df.Country.isin(df2.Type))] #Find same elements
>>> df3.filter(items="a","b"]) #Filter on values
>>> df.select(lambda x: not x%5) #Select specific elements
```

Where

```
>>> s.where(s > 0) #Subset the data
```

Query

```
>>> df6.query('second > first') #Query DataFrame
```

Setting/Resetting Index

```
>>> df.set_index('Country') #Set the index
>>> df4 = df.reset_index() #Reset the index
>>> df = df.rename(index=str, #Rename
                   DataFrame columns={"Country":"cntry",
                                      "Capital":"cptl",
                                      "Population":"ppltn"})
```

Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

```
   Country    Capital      Population
0 Belgium    Brussels     11190846
1 India      New Delhi    1303171035
2 Brazil     Brasília     207847528
3 Brazil     Brasília     207847528
```

Backward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

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

MultiIndexing

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

> Duplicate Data

```
>>> s3.unique() #Return unique values
>>> df2.duplicated('Type') #Check duplicates
>>> df2.drop_duplicates('Type', keep='last') #Drop duplicates
>>> df.index.duplicated() #Check index duplicates
```

> Grouping Data

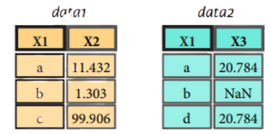
Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x), 'b': np.sum})
```

Transformation

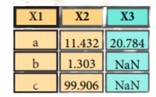
```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

> Combining Data

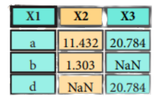


Merge

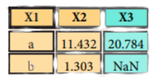
```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```



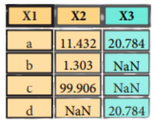
```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```



```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```



```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```



Join

```
>>> data1.join(data2, how='right')
```

Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

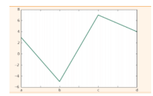
> Dates

```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

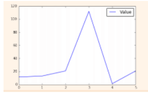
> Visualization

Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> s.plot()
>>> plt.show()
```



```
>>> df2.plot()
>>> plt.show()
```



datacamp

Python For Data Science

SciPy Cheat Sheet

Learn SciPy online at www.DataCamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

> Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5] #Create a dense meshgrid
>>> np.ogrid[0:2,0:2] #Create an open meshgrid
>>> np.r_[[3,[0]*5,-1:1:10j] #Stack arrays vertically (row-wise)
>>> np.c_[b,c] #Create stacked column-wise arrays
```

Shape Manipulation

```
>>> np.transpose(b) #Permute array dimensions
>>> b.flatten() #Flatten the array
>>> np.hstack((b,c)) #Stack arrays horizontally (column-wise)
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
>>> np.hsplit(c,2) #Split the array horizontally at the 2nd index
>>> np.vpslit(d,2) #Split the array vertically at the 2nd index
```

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5]) #Create a polynomial object
```

Vectorizing Functions

```
>>> def myfunc(a): if a < 0:
        return a*2
        else:
        return a/2
>>> np.vectorize(myfunc) #Vectorize functions
```

Type Handling

```
>>> np.real(c) #Return the real part of the array elements
>>> np.imag(c) #Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000) #Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi) #Cast object to a data type
```

Other Useful Functions

```
>>> np.angle(b,deg=True) #Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5) #Create an array of evenly spaced values(number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g) #Unwrap
>>> np.logspace(0,10,3) #Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2]) #Return values from a list of arrays depending on conditions
>>> misc.factorial(a) #Factorial
>>> misc.comb(10,3,exact=True) #Combine N things taken at k time
>>> misc.central_diff_weights(3) #Weights for Np-point central derivative
>>> misc.derivative(myfunc,1.0) #Find the n-th derivative of a function at a point
```

> Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules.
Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I #Inverse
>>> linalg.inv(A) #Inverse
>>> A.T #Tranpose matrix
>>> A.H #Conjugate transposition
>>> np.trace(A) #Trace
```

Norm

```
>>> linalg.norm(A) #Frobenius norm
>>> linalg.norm(A,1) #L1 norm (max column sum)
>>> linalg.norm(A,np.inf) #L inf norm (max row sum)
```

Rank

```
>>> np.linalg.matrix_rank(C) #Matrix rank
```

Determinant

```
>>> linalg.det(A) #Determinant
```

Solving linear problems

```
>>> linalg.solve(A,b) #Solver for dense matrices
>>> E = np.mat(a).T #Solver for dense matrices
>>> linalg.lstsq(D,E) #Least-squares solution to linear matrix equation
```

Generalized inverse

```
>>> linalg.pinv(C) #Compute the pseudo-inverse of a matrix (least-squares solver)
>>> linalg.pinv2(C) #Compute the pseudo-inverse of a matrix  (SVD)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1) #Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2)) #Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C) #Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D) #Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A) #Dictionary Of Keys matrix
>>> E.todense() #Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A) #Identify sparse matrix
```

Sparse Matrix Routines

Inverse

```
>>> sparse.linalg.inv(I) #Inverse
```

Norm

```
>>> sparse.linalg.norm(I) #Norm
```

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I) #Solver for sparse matrices
```

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I) #Sparse matrix exponential
```

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1) #Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2) #SVD
```

> Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Matrix Functions

Addition

```
>>> np.add(A,D) #Addition
```

Subtraction

```
>>> np.subtract(A,D) #Subtraction
```

Division

```
>>> np.divide(A,D) #Division
```

Multiplication

```
>>> np.multiply(D,A) #Multiplication
>>> np.dot(A,D) #Dot product
>>> np.vdot(A,D) #Vector dot product
>>> np.inner(A,D) #Inner product
>>> np.outer(A,D) #Outer product
>>> np.tensordot(A,D) #Tensor dot product
>>> np.kron(A,D) #Kronecker product
```

Exponential Functions

```
>>> linalg.expm(A) #Matrix exponential
>>> linalg.expm2(A) #Matrix exponential (Taylor Series)
>>> linalg.expm3(D) #Matrix exponential (eigenvalue decomposition)
```

Logarithm Function

```
>>> linalg.logm(A) #Matrix logarithm
```

Trigonometric Functions

```
>>> linalg.sinm(D) Matrix sine
>>> linalg.cosm(D) Matrix cosine
>>> linalg.tanm(A) Matrix tangent
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D) #Hypberbolic matrix sine
>>> linalg.coshm(D) #Hyperbolic matrix cosine
>>> linalg.tanhm(A) #Hyperbolic matrix tangent
```

Matrix Sign Function

```
>>> np.sigm(A) #Matrix sign function
```

Matrix Square Root

```
>>> linalg.sqrtm(A) #Matrix square root
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x) #Evaluate matrix function
```

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A) #Solve ordinary or generalized eigenvalue problem for square matrix
>>> l1, l2 = la #Unpack eigenvalues
>>> v[:,0] #First eigenvector
>>> v[:,1] #Second eigenvector
>>> linalg.eigvals(A) #Unpack eigenvalues
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B) #Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N) #Construct sigma matrix in SVD
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C) #LU Decomposition
```

datacamp

Python For Data Science
NumPy Cheat Sheet

Learn NumPy online at www.DataCamp.com

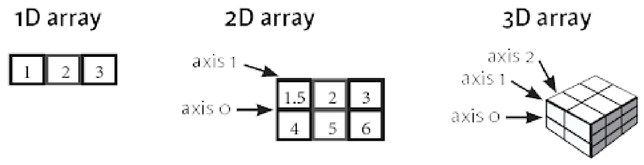
Numpy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



> Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)],[(3,2,1), (4,5,6)]], dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4)) #Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) #Create an array of ones
>>> d = np.arange(10,25,5) #Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9) #Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7) #Create a constant array
>>> f = np.eye(2) #Create a 2X2 identity matrix
>>> np.random.random((2,2)) #Create an array with random values
>>> np.empty((3,2)) #Create an empty array
```

> I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

> Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

> Inspecting Your Array

```
>>> a.shape #Array dimensions
>>> len(a) #Length of array
>>> b.ndim #Number of array dimensions
>>> e.size #Number of array elements
>>> b.dtype #Data type of array elements
>>> b.dtype.name #Name of data type
>>> b.astype(int) #Convert an array to a different type
```

> Data Types

```
>>> np.int64 #Signed 64-bit integer types
>>> np.float32 #Standard double-precision floating point
>>> np.complex #Complex numbers represented by 128 floats
>>> np.bool #Boolean type storing TRUE and FALSE values
>>> np.object #Python object type
>>> np.string_ #Fixed-length string type
>>> np.unicode_ #Fixed-length unicode type
```

> Array Mathematics

Arithmetic Operations

```
>>> g = a - b #Subtraction
 array([[-0.5, 0. , 0. ],
        [-3. , -3. , -3. ]])
>>> np.subtract(a,b) #Subtraction
>>> b + a #Addition
 array([[ 2.5, 4. , 6. ],
        [ 5. , 7. , 9. ]])
>>> np.add(b,a) #Addition
>>> a / b #Division
 array([[ 0.66666667, 1. , 1. ],
        [ 0.25 , 0.4 , 0.5 ]])
>>> np.divide(a,b) #Division
>>> a * b #Multiplication
 array([[ 1.5, 4. , 9. ],
        [ 4. , 10. , 18. ]])
>>> np.multiply(a,b) #Multiplication
>>> np.exp(b) #Exponentiation
>>> np.sqrt(b) #Square root
>>> np.sin(a) #Print sines of an array
>>> np.cos(b) #Element-wise cosine
>>> np.log(a) #Element-wise natural logarithm
>>> e.dot(f) #Dot product
 array([[ 7., 7.],
        [ 7., 7.]])
```

Comparison

```
>>> a == b #Element-wise comparison
 array([[False, True, True],
        [False, False, False]], dtype=bool)
>>> a < 2 #Element-wise comparison
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b) #Array-wise comparison
```

Aggregate Functions

```
>>> a.sum() #Array-wise sum
>>> a.min() #Array-wise minimum value
>>> b.max(axis=0) #Maximum value of an array row
>>> b.cumsum(axis=1) #Cumulative sum of the elements
>>> a.mean() #Mean
>>> np.median(b) #Median
>>> np.corrcoef(a) #Correlation coefficient
>>> np.std(b) #Standard deviation
```

> Copying Arrays

```
>>> h = a.view() #Create a view of the array with the same data
>>> np.copy(a) #Create a copy of the array
>>> h = a.copy() #Create a deep copy of the array
```

> Sorting Arrays

```
>>> a.sort() #Sort an array
>>> c.sort(axis=0) #Sort the elements of an array's axis
```

> Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2] #Select the element at the 2nd index
 3
>>> b[1,2] #Select the element at row 1 column 2 (equivalent to b[1][2])
 6.0
```

Slicing

```
>>> a[0:2] #Select items at index 0 and 1
 array([1, 2])
>>> b[0:2,1] #Select items at rows 0 and 1 in column 1
 array([ 2., 5.])
>>> b[:1] #Select all items at row 0 (equivalent to b[0:1, :])
 array([[1.5, 2., 3.]])
>>> c[1,...] #Same as [1,:,:]
 array([[[ 3., 2., 1.],
         [ 4., 5., 6.]]])
>>> a[ : :-1] #Reversed array a array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2] #Select elements from a less than 2
 array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]] #Select elements (1,0),(0,1),(1,2) and (0,0)
 array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]] #Select a subset of the matrix's rows and columns
 array([[ 4. ,5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5],
        [ 4. , 5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5]])
```

> Array Manipulation

Transposing Array

```
>>> i = np.transpose(b) #Permute array dimensions
>>> i.T #Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel() #Flatten the array
>>> g.reshape(3,-2) #Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6)) #Return a new array with shape (2,6)
>>> np.append(h,g) #Append items to an array
>>> np.insert(a, 1, 5) #Insert items in an array
>>> np.delete(a,[1]) #Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0) #Concatenate arrays
 array([ 1, 2, 3, 10, 15, 20])
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
 array([[ 1. , 2. , 3. ],
        [ 1.5, 2. , 3. ],
        [ 4. , 5. , 6. ]])
>>> np.r_[e,f] #Stack arrays vertically (row-wise)
>>> np.hstack((e,f)) #Stack arrays horizontally (column-wise)
 array([[ 7., 7., 1., 0.],
        [ 7., 7., 0., 1.]])
>>> np.column_stack((a,d)) #Create stacked column-wise arrays
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d] #Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3) #Split the array horizontally at the 3rd index
 [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2) #Split the array vertically at the 2nd index
 [array([[[ 1.5, 2. , 1. ],
          [ 4. , 5. , 6. ]]]),
 array([[[ 3., 2., 3.],
         [ 4., 5., 6.]]])]
```

datacamp

Python For Data Science
spaCy Cheat Sheet

Learn spaCy online at www.DataCamp.com

spaCy

spaCy is a free, open-source library for advanced Natural Language processing (NLP) in Python. It's designed specifically for production use and helps you build applications that process and "understand" large volumes of text. Documentation: spacy.io

```
>>> $ pip install spacy
>>> import spacy
```

> Statistical models

Download statistical models

Predict part-of-speech tags, dependency labels, named entities and more. See here for available models: spacy.io/models

```
>>> $ python -m spacy download en_core_web_sm
```

Check that your installed models are up to date

```
>>> $ python -m spacy validate
```

Loading statistical models

```
>>> import spacy
>>> nlp = spacy.load("en_core_web_sm") # Load the installed model "en_core_web_sm"
```

> Documents and tokens

Processing text

Processing text with the `nlp` object returns a `Doc` object that holds all information about the tokens, their linguistic features and their relationships

```
>>> doc = nlp("This is a text")
```

Accessing token attributes

```
>>> doc = nlp("This is a text")
>>>[token.text for token in doc] #Token texts
 ['This', 'is', 'a', 'text']
```

> Label explanations

```
>>> spacy.explain("RB")
 'adverb'
>>> spacy.explain("GPE")
 'Countries, cities, states'
```

> Spans

Accessing spans

Span indices are exclusive. So `doc[2:4]` is a span starting at token 2, up to – but not including! – token 4.

```
>>> doc = nlp("This is a text")
>>> span = doc[2:4]
>>> span.text
 'a text'
```

Creating a span manually

```
>>> from spacy.tokens import Span #Import the Span object
>>> doc = nlp("I live in New York") #Create a Doc object
>>> span = Span(doc, 3, 5, label="GPE") #Span for "New York" with label GPE (geopolitical)
>>> span.text
 'New York'
```

> Linguistic features

Attributes return label IDs. For string labels, use the attributes with an underscore. For example, `token.pos_` .

Part-of-speech tags

Predicted by Statistical model

```
>>> doc = nlp("This is a text.")
>>> [token.pos_ for token in doc] #Coarse-grained part-of-speech tags
 ['DET', 'VERB', 'DET', 'NOUN', 'PUNCT']
>>> [token.tag_ for token in doc] #Fine-grained part-of-speech tags
 ['DT', 'VBZ', 'DT', 'NN', '.']
```

Syntactic dependencies

Predicted by Statistical model

```
>>> doc = nlp("This is a text.")
>>> [token.dep_ for token in doc] #Dependency labels
 ['nsubj', 'ROOT', 'det', 'attr', 'punct']
>>> [token.head.text for token in doc] #Syntactic head token (governor)
 ['is', 'is', 'text', 'is', 'is']
```

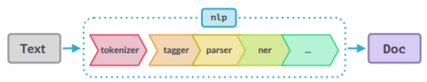
Named entities

Predicted by Statistical model

```
>>> doc = nlp("Larry Page founded Google")
>>> [(ent.text, ent.label_) for ent in doc.ents] #Text and label of named entity span
 [('Larry Page', 'PERSON'), ('Google', 'ORG')]
```

> Pipeline components

Functions that take a `Doc` object, modify it and return it.



Pipeline information

```
>>> nlp = spacy.load("en_core_web_sm")
>>> nlp.pipe_names
 ['tagger', 'parser', 'ner']
>>> nlp.pipeline
 [('tagger', <spacy.pipeline.Tagger>),
 ('parser', <spacy.pipeline.DependencyParser>),
 ('ner', <spacy.pipeline.EntityRecognizer>)]
```

Custom components

```
def custom_component(doc): #Function that modifies the doc and returns it
    print("Do something to the doc here!")
    return doc
nlp.add_pipe(custom_component, first=True) #Add the component first in the pipeline
```

Components can be added `first` , `last` (default), or before or after an existing component.

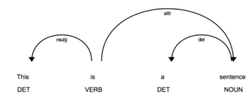
> Visualizing

If you're in a Jupyter notebook, use `displacy.render` otherwise, use `displacy.serve` to start a web server and show the visualization in your browser.

```
>>> from spacy import displacy
```

Visualize dependencies

```
>>> doc = nlp("This is a sentence")
>>> displacy.render(doc, style="dep")
```



Visualize named entities

```
>>> doc = nlp("Larry Page founded Google")
>>> displacy.render(doc, style="ent")
```

Larry Page PERSON founded Google ORG

> Word vectors and similarity

To use word vectors, you need to install the larger models ending in `md` or `lg` , for example `en_core_web_lg` .

Comparing similarity

```
>>> doc1 = nlp("I like cats")
>>> doc2 = nlp("I like dogs")
>>> doc1.similarity(doc2) #Compare 2 documents
>>> doc1[2].similarity(doc2[2]) #Compare 2 tokens
>>> doc1[0].similarity(doc2[1:3]) # Comparetokens and spans
```

Accessing word vectors

```
>>> doc = nlp("I like cats") #Vector as a numpy array
>>> doc[2].vector #The L2 norm of the token's vector
>>> doc[2].vector_norm
```

> Syntax iterators

Sentences

Ususally needs the dependency parser

```
>>> doc = nlp("This a sentence. This is another one.")
>>> [sent.text for sent in doc.sents] #doc.sents is a generator that yields sentence spans
 ['This is a sentence.', 'This is another one.']
```

Base noun phrases

Needs the tagger and parser

```
>>> doc = nlp("I have a red car")
#doc.noun_chunks is a generator that yields spans
>>> [chunk.text for chunk in doc.noun_chunks]
 ['I', 'a red car']
```

> Extension attributes

Custom attributes that are registered on the global `Doc`, `Token` and `Span` classes and become available as `._`.

```
>>> from spacy.tokens import Doc, Token, Span
>>> doc = nlp("The sky over New York is blue")
```

Attribute extensions — With default value

```
# Register custom attribute on Token class
>>> Token.set_extension("is_color", default=False)
# Overwrite extension attribute with default value
doc[6]._.is_color = True
```

Property extensions — With getter and setter

```
# Register custom attribute on Doc class
>>> get_reversed = lambda doc: doc.text[::-1]
>>> Doc.set_extension("reversed", getter=get_reversed)
# Compute value of extension attribute with getter
>>> doc._.reversed
'eulb si kroY weN revo yks ehT'
```

Method extensions — Callable Method

```
# Register custom attribute on Span class
>>> has_label = lambda span, label: span.label_ == label
>>> Span.set_extension("has_label", method=has_label)
# Compute value of extension attribute with method
>>> doc[3:5].has_label("GPE")
True
```

> Rule-based matching

Using the matcher

```
# Matcher is initialized with the shared vocab
>>> from spacy.matcher import Matcher
# Each dict represents one token and its attributes
>>> matcher = Matcher(nlp.vocab)
# Add with ID, optional callback and pattern(s)
>>> pattern = [{"LOWER": "new"}, {"LOWER": "york"}]
>>> matcher.add("CITIES", None, pattern)
# Match by calling the matcher on a Doc object
>>> doc = nlp("I live in New York")
>>> matches = matcher(doc)
# Matches are (match_id, start, end) tuples
>>> for match_id, start, end in matches:
    # Get the matched span by slicing the Doc
    span = doc[start:end]
    print(span.text)
    'New York'
```

Token patterns

```
# "love cats", "loving cats", "loved cats"
>>> pattern1 = [{"LEMMA": "love"}, {"LOWER": "cats"}]
# "10 people", "twenty people"
>>> pattern2 = [{"LIKE_NUM": True}, {"TEXT": "people"}]
# "book", "a cat", "the sea" (noun + optional article)
>>> pattern3 = [{"POS": "DET", "OP": "?"}, {"POS": "NOUN"}]
```

Operators and quantifiers

Can be added to a token dict as the `"OP"` key

`!`	Negate pattern and match exactly 0 times
`?`	Make pattern optional and match 0 or 1 times
`+`	Require pattern to match 1 or more times
`*`	Allow pattern to match 0 or more time

> Glossary

Tokenization

Segmenting text into words, punctuation etc

Lemmatization

Assigning the base forms of words, for example:
"was" → "be" or "rats" → "rat".

Sentence Boundary Detection

Finding and segmenting individual sentences.

Part-of-speech (POS) Tagging

Assigning word types to tokens like verb or noun.

Dependency Parsing

Assigning syntactic dependency labels,
describing the relations between individual
tokens, like subject or object.

Named Entity Recognition (NER)

Labeling named "real-world" objects,
like persons, companies or locations.

Text Classification

Assigning categories or labels to a whole
document, or parts of a document.

Statistical model

Process for making predictions based on examples.

Training

Updating a statistical model with new examples.

datacamp

Python For Data Science
Scikit-Learn Cheat Sheet

Learn Scikit-Learn online at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

> Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

> Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

> Model Fitting

Supervised learning

```
>>> lr.fit(X, y) #Fit the model to the data
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Unsupervised Learning

```
>>> k_means.fit(X_train) #Fit the model to the data
>>> pca_model = pca.fit_transform(X_train) #Fit to data, then transform it
```

> Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5))) #Predict labels
>>> y_pred = lr.predict(X_test) #Predict labels
>>> y_pred = knn.predict_proba(X_test) #Estimate probability of a label
```

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test) #Predict labels in clustering algos
```

> Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

> Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

> Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test) #Estimator score method
>>> from sklearn.metrics import accuracy_score #Metric scoring functions
>>> accuracy_score(y_test, y_pred)
```

Classification Report

```
>>> from sklearn.metrics import classification_report #Precision, recall, f1-score and support
>>> print(classification_report(y_test, y_pred))
```

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

> Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5), "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn, param_distributions=params,
                                 cv=4, n_iter=8, random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

datacamp

Python For Data Science

PySpark SQL Basics Cheat Sheet

Learn PySpark SQL online at www.DataCamp.com

PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

> Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

> Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
```

Infer Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

Specify Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
```

```
+--------+---+
|    name|age|
+--------+---+
|    Mine| 28|
|   Filip| 29|
|Jonathan| 30|
+--------+---+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
```

```
+--------------------+---+---------+--------+--------------------+
|             address|age|firstName |lastName|         phoneNumber|
+--------------------+---+---------+--------+--------------------+
|[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
|[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
+--------------------+---+---------+--------+--------------------+
```

```
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet files

```
>>> df3 = spark.read.load("users.parquet")
```

TXT files

```
>>> df4 = spark.read.text("people.txt")
```

> Filter

```
#Filter entries of age, only keep those records of which the values are >24
>>> df.filter(df["age"]>24).show()
```

> Duplicate Values

```
>>> df = df.dropDuplicates()
```

> Queries

```
>>> from pyspark.sql import functions as F
```

Select

```
>>> df.select("firstName").show() #Show all entries in firstName column
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName", #Show all entries in firstName, age and type
              "age",
              explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
              "firstName",
              "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)  #Show all entries in firstName and age,
      .show()                                  add 1 to the entries of age
>>> df.select(df['age'] > 24).show() #Show all entries where age >24
```

When

```
>>> df.select("firstName", #Show firstName and 0 or 1 depending on age >30
              F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")] #Show firstName if in the given options
                 .collect()
```

Like

```
>>> df.select("firstName", #Show firstName, and lastName is TRUE if lastName is like Smith
              df.lastName.like("Smith")) \
      .show()
```

Startswith - Endswith

```
>>> df.select("firstName", #Show firstName, and TRUE if lastName starts with Sm
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th"))\ #Show last names ending in th
      .show()
```

Substring

```
>>> df.select(df.firstName.substr(1, 3) \ #Return substrings of firstName
                          .alias("name")) \
      .collect()
```

Between

```
>>> df.select(df.age.between(22, 24)) \ #Show age: values are TRUE if between 22 and 24
      .show()
```

> Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
           .withColumn('postalCode',df.address.postalCode) \
           .withColumn('state',df.address.state) \
           .withColumn('streetAddress',df.address.streetAddress) \
           .withColumn('telePhoneNumber', explode(df.phoneNumber.number)) \
           .withColumn('telePhoneType', explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

> Missing & Replacing Values

```
>>> df.na.fill(50).show() #Replace null values
>>> df.na.drop().show() #Return new df omitting rows with null values
>>> df.na \ #Return new df replacing one value with another
      .replace(10, 20) \
      .show()
```

> GroupBy

```
>>> df.groupBy("age")\ #Group by age, count the members in the groups
      .count() \
      .show()
```

> Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
      .collect()
```

> Repartitioning

```
>>> df.repartition(10)\ #df with 10 partitions
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions() #df with 1 partition
```

> Running Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                 .show()
```

> Inspect Data

```
>>> df.dtypes #Return df column names and data types
>>> df.show() #Display the content of df
>>> df.head() #Return first n rows
>>> df.first() #Return first row
>>> df.take(2) #Return the first n rows >>> df.schema Return the schema of df
>>> df.describe().show() #Compute summary statistics >>> df.columns Return the columns of df
>>> df.count() #Count the number of rows in df
>>> df.distinct().count() #Count the number of distinct rows in df
>>> df.printSchema() #Print the schema of df
>>> df.explain() #Print the (logical and physical) plans
```

> Output

Data Structures

```
>>> rdd1 = df.rdd #Convert df into an RDD
>>> df.toJSON().first() #Convert df into a RDD of string
>>> df.toPandas() #Return the contents of df as Pandas DataFrame
```

Write & Save to Files

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json",format="json")
```

> Stopping SparkSession

```
>>> spark.stop()
```

datacamp

Python For Data Science

PySpark RDD Cheat Sheet

Learn PySpark RDD online at www.DataCamp.com

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

> Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version #Retrieve SparkContext version
>>> sc.pythonVer #Retrieve Python version
>>> sc.master #Master URL to connect to
>>> str(sc.sparkHome) #Path where Spark is installed on worker nodes
>>> str(sc.sparkUser()) #Retrieve name of the Spark User running SparkContext
>>> sc.appName #Return application name
>>> sc.applicationId #Retrieve application ID
>>> sc.defaultParallelism #Return default level of parallelism
>>> sc.defaultMinPartitions #Default minimum number of partitions for RDDs
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

> Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

> Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions() #List the number of partitions
>>> rdd.count() #Count RDD instances 3
>>> rdd.countByKey() #Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue() #Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap() #Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum() #Sum of RDD elements 4950
>>> sc.parallelize([]).isEmpty() #Check whether RDD is empty
 True
```

Summary

```
>>> rdd3.max() #Maximum value of RDD elements
 99
>>> rdd3.min() #Minimum value of RDD elements
 0
>>> rdd3.mean() #Mean value of RDD elements
 49.5
>>> rdd3.stdev() #Standard deviation of RDD elements
 28.866070047722118
>>> rdd3.variance() #Compute variance of RDD elements
 833.25
>>> rdd3.histogram(3) #Compute histogram by bins
 ([0,33,66,99],[33,33,34])
>>> rdd3.stats() #Summary statistics (count, mean, stdev, max & min)
```

> Applying Functions

```
#Apply a function to each RDD element
>>> rdd.map(lambda x: x+(x[1],x[0])).collect()
 [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
#Apply a function to each RDD element and flatten the result
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
>>> rdd5.collect()
 ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
#Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
>>> rdd4.flatMapValues(lambda x: x).collect()
 [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

> Selecting Data

Getting

```
>>> rdd.collect() #Return a list with all RDD elements
 [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2) #Take first 2 RDD elements
 [('a', 7), ('a', 2)]
>>> rdd.first() #Take first RDD element
 ('a', 7)
>>> rdd.top(2) #Take top 2 RDD elements
 [('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect() #Return sampled subset of rdd3
 [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x).collect()  #Filter the RDD
 [('a',7),('a',2)]
>>> rdd5.distinct().collect() #Return distinct RDD values
 ['a',2,'b',7]
>>> rdd.keys().collect() #Return (key,value) RDD's keys
 ['a', 'a', 'b']
```

> Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g) #Apply a function to all RDD elements
 ('a', 7)
 ('b', 2)
 ('a', 2)
```

> Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y).collect() #Merge the rdd values for each key
 [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b) #Merge the rdd values
 ('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2) #Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey() #Group rdd by key
       .mapValues(list)
       .collect()
 [('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
#Aggregate RDD elements of each partition and then the results
>>> rdd3.aggregate((0,0),seqOp,combOp)
 (4950,100)
 #Aggregate values of each RDD key
>>> rdd.aggregateByKey((0,0),seqop,combop).collect()
 [('a',(9,2)), ('b',(2,1))]
#Aggregate the elements of each partition, and then the results
>>> rdd3.fold(0,add)
 4950
#Merge the values for each key
>>> rdd.foldByKey(0, add).collect()
 [('a',9),('b',2)]
#Create tuples of RDD elements by applying a function
>>> rdd3.keyBy(lambda x: x+x).collect()
```

> Mathematical Operations

```
>>> rdd.subtract(rdd2).collect() #Return each rdd value not contained in rdd2
 [('b',2),('a',7)]
#Return each (key,value) pair of rdd2  with no matching key in rdd
>>> rdd2.subtractByKey(rdd).collect()
 [('d', 1)]
>>> rdd.cartesian(rdd2).collect() #Return the Cartesian product of rdd and rdd2
```

> Sort

```
>>> rdd2.sortBy(lambda x: x[1]).collect() #Sort RDD by given function
 [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey().collect() #Sort (key, value) RDD by key
 [('a',2),('b',1),('d',1)]
```

> Repartitioning

```
>>> rdd.repartition(4) #New RDD with 4 partitions
>>> rdd.coalesce(1) #Decrease the number of partitions in the RDD to 1
```

> Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

> Stopping SparkContext

```
>>> sc.stop()
```

> Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

datacamp

Python For Data Science

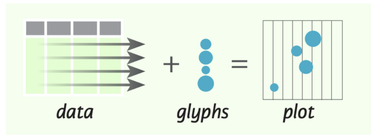
Bokeh Cheat Sheet

Learn Bokeh online at www.DataCamp.com
taught by Bryan Van de Ven, core contributor

Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data (Python lists, NumPy arrays, Pandas DataFrames and other sequences of values)
2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5] #Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example", #Step 2
 x_axis_label='x',
 y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2) #Step 3
>>> output_file("lines.html") #Step 4
>>> show(p) #Step 5
```

1 Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

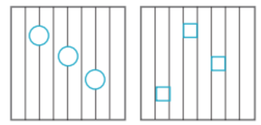
```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4, 4, 66, 'Asia'],
                                [21.4, 4, 109, 'Europe']]),
                   columns=['mpg','cyl', 'hp', 'origin'],
                   index=['Toyota', 'Fiat', 'Volvo'])

>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
               x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

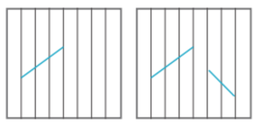
3 Renderers & Visual Customizations

Glyphs



Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```



Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> from bokeh.models import HoverTool
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

Colormapping

```
>>> from bokeh.models import CategoricalColorMapper
>>> color_mapper = CategoricalColorMapper(
                   factors=['US', 'Asia', 'Europe'],
                   palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
              color=dict(field='origin',
                         transform=color_mapper),
              legend='Origin')
```

4 Output & Export

Notebook

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

HTML

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> from bokeh.resources import CDN
>>> html = file_html(p, CDN, "my_plot")

>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> from bokeh.models import Legend
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" ,[p1, r1]),("Two",[r2])],
                    location=(0, -30))
>>> p.add_layout(legend, 'right')
```

Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

Columns

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

Nesting Rows & Columns

```
>>>layout = row(column(p1,p2), p3)
```

Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Linked Plots

Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100, tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200, tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

5 Show or Save Your Plots

```
>>> show(p1)
>>> show(layout)
>>> save(p1)
>>> save(layout)
```

datacamp

Python For Data Science
Seaborn Cheat Sheet

Learn Seaborn online at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot
5. Show your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips") #Step 1
>>> sns.set_style("whitegrid") #Step 2
>>> g = sns.lmplot(x="tip", #Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))
>>> plt.title("title") #Step 4
>>> plt.show(g) #Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6)) #Create a figure and one subplot
```

Seaborn styles

```
>>> sns.set() #(Re)set the seaborn default
>>> sns.set_style("whitegrid") #Set the matplotlib parameters
>>> sns.set_style("ticks", #Set the matplotlib parameters
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
#Return a dict of params or use with with to temporarily set the style
>>> sns.axes_style("whitegrid")
```

Context Functions

```
>>> sns.set_context("talk") #Set context to "talk"
>>> sns.set_context("notebook", #Set context to "notebook",
                    font_scale=1.5, #Scale font elements and
                    rc={"lines.linewidth":2.5}) #override param mapping
```

Color Palette

```
>>> sns.set_palette("husl",3) #Define the color palette
>>> sns.color_palette("husl") #Use with with to temporarily set palette
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui) #Set your own color palette
```

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic, #Subplot grid for plotting conditional relationships
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
>>> sns.factorplot(x="pclass", #Draw a categorical plot onto a Facetgrid
                   y="survived",
                   hue="sex",
                   data=titanic)
>>> sns.lmplot(x="sepal_width", #Plot data and regression model fits across a FacetGrid
               y="sepal_length",
               hue="species",
               data=iris)
>>> h = sns.PairGrid(iris) #Subplot grid for plotting pairwise relationships
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris) #Plot pairwise bivariate distributions
>>> i = sns.JointGrid(x="x", #Grid for bivariate plot with marginal univariate plots
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
>>> sns.jointplot("sepal_length", #Plot bivariate distribution
                  "sepal_width",
                  data=iris,
                  kind='kde')
```

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True) #Remove left spine
>>> g.set_ylabels("Survived") #Set the labels of the y-axis
>>> g.set_xticklabels(rotation=45) #Set the tick labels for x
>>> g.set_axis_labels("Survived", #Set the axis labels
                      "Sex")
>>> h.set(xlim=(0,5), #Set the limit and ticks of the x-and y-axis
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```

Plot

```
>>> plt.title("A Title") #Add plot title
>>> plt.ylabel("Survived") #Adjust the label of the y-axis
>>> plt.xlabel("Sex") #Adjust the label of the x-axis
>>> plt.ylim(0,100) #Adjust the limits of the y-axis
>>> plt.xlim(0,10) #Adjust the limits of the x-axis
>>> plt.setp(ax,yticks=[0,5]) #Adjust a plot property
>>> plt.tight_layout() #Adjust subplot params
```

Regression Plots

```
>>> sns.regplot(x="sepal_width", #Plot data and a linear regression model fit
                y="sepal_length",
                data=iris,
                ax=ax)
```

Distribution Plots

```
>>> plot = sns.distplot(data.y, #Plot univariate distribution
                        kde=False,
                        color="b")
```

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1) #Heatmap
```

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species", #Scatterplot with one categorical variable
                  y="petal_length",
                  data=iris)
>>> sns.swarmplot(x="species", #Categorical scatterplot with non-overlapping points
                  y="petal_length",
                  data=iris)
```

Bar Chart

```
>>> sns.barplot(x="sex", #Show point estimates & confidence intervals with scatterplot glyphs
                y="survived",
                hue="class",
                data=titanic)
```

Count Plot

```
>>> sns.countplot(x="deck", #Show count of observations
                  data=titanic,
                  palette="Greens_d")
```

Point Plot

```
>>> sns.pointplot(x="class", #Show point estimates & confidence intervals as rectangular bars
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                  "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```

Boxplot

```
>>> sns.boxplot(x="alive", #Boxplot
                y="age",
                hue="adult_male",
                data=titanic)
>>> sns.boxplot(data=iris,orient="h") #Boxplot with wide-form data
```

Violinplot

```
>>> sns.violinplot(x="age", #Violin plot
                   y="sex",
                   hue="survived",
                   data=titanic)
```

5 Show or Save Plot

Also see Matplotlib

```
>>> plt.show() #Show the plot
>>> plt.savefig("foo.png") #Save the plot as a figure
>>> plt.savefig("foo.png", #Save transparent figure
                transparent=True)
```

> Close & Clear

Also see Matplotlib

```
>>> plt.cla() #Clear an axis
>>> plt.clf() #Clear an entire figure
>>> plt.close() #Close a window
```

datacamp

Python For Data Science
Keras Cheat Sheet

Learn Keras online at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library forTheano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

> Data

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing, mnist, cifar10, imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data =
np.loadtxt(urlopen("http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

> Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

> Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

> Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X, y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

> Inspect Model

```
>>> model.output_shape #Model output shape
>>> model.summary() #Model summary representation
>>> model.get_config() #Model configuration
>>> model.get_weights() #List all weight tensors in the model
```

> Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

> Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

> Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

> Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

> Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

datacamp

Python For Data Science

Matplotlib Cheat Sheet

Learn Matplotlib online at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

> Prepare The Data

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

> Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs.
A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) #row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

> Save Plot

```
>>> plt.savefig('foo.png')  #Save figures
>>> plt.savefig('foo.png', transparent=True)  #Save transparent figures
```

> Show Plot

```
>>> plt.show()
```

> Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y) #Draw points with lines or markers connecting them
>>> ax.scatter(x,y) #Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5]) #Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2]) #Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45) #Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65) #Draw a vertical line across axes
>>> ax.fill(x,y,color='blue') #Draw filled polygons
>>> ax.fill_between(x,y,color='yellow') #Fill between y-values and 0
```

2D Data

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img, #Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
>>> axes2[0].pcolor(data2) #Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data) #Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U) #Plot contours
>>> axes2[2].contourf(data1) #Plot filled contours
>>> axes2[2]= ax.clabel(CS) #Label a contour plot
```

Vector Fields

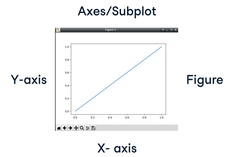
```
>>> axes[0,1].arrow(0,0,0.5,0.5) #Add an arrow to the axes
>>> axes[1,1].quiver(y,z) #Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V) #Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y) #Plot a histogram
>>> ax3.boxplot(y) #Make a box and whisker plot
>>> ax3.violinplot(z) #Make a violin plot
```

> Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare Data 2 Create Plot 3 Plot 4 Customized Plot 5 Save Plot 6 Show Plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4] #Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure() #Step 2
>>> ax = fig.add_subplot(111) #Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3) #Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png') #Step 5
>>> plt.show() #Step 6
```

> Close and Clear

```
>>> plt.cla() #Clear an axis
>>> plt.clf() #Clear the entire figure
>>> plt.close() #Close a window
```

> Plotting Cutomize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends and Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1) #Add padding to a plot
>>> ax.axis('equal') #Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5]) #Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5) #Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes', #Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best') #No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5), #Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y', #Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5, #Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout() #Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)  #Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10)) #Move the bottom axis line outward
```