Cheatography

Python Basics Cheat Sheet

by LasseBoe (Lasse1618) via cheatography.com/110079/cs/21547/

Basic Operations

type(x)	int/float/bool/str/co-mplex
bin(x)	binary
hex(x)	hexadecimal
1e8	$1*10^8$
inf	infinity
NaN	not a number
abs(x)	absolute value
x//3	floor division
x *= 2	x = x*2
3 % 2	remainder
a ** x	a^x
&	and
|	or
^	xor
==	check for equality (->bool)
!=	not equal
range(start, finish, stepsize)	includes start, excludes finish

files

f = open('file.txt', 'w')	opens the file 'w' = write&read 'r' = read
f.write('This is a file')	
f.open	
f.close	
f.flush	can be removed safely now
f.readline()	reads first line
f.readline(x)	reads first x positions
f = open(*<path>*, 'w') for line in f: code f.close()	

control structures

if x = y:
command
elif:
else:
	if condition

break	ends innermost loop
if x=y and y<z:	two conditions
for i in range(10):	for loop
while i < 5:	while loop
my_iter = my_list.__iter__()	creates an iterator
next(my_iter) *my_iter.__ne-xt__()*	gets next element from iter

vectors & matrices

import numpy as np	
v = np.array([[3, 2]]) note: np.array([3, 2]) is a rank 1 array, NOT a math. vector	Vectors are lists of lists
v.T	transpose
v.shape	returns dimensions
np.linalg.norm(v){{nl]]*or* math.sqrt(dot(v.T,v))	length of vector
A = np.array([[1,2], [3,4]])	creates a matrix
np.random.rand(2,2)	random matrix values normal distribution 0 to 1
np.random.randn(2,2)	normal distri-bution (mean 0, variance 1)
A.dtype	specifically for np
A.ndim	number of dimensions

vectors & matrices (cont)

np.arange(x)	range as array with x elements
np.identity(x)	identity matrix
np.zeros(-(x,y))	zeros matrix
A[0]	first row
A[0][0] *or* A[0, 0]	first element of first row
A[:, 0]	first column
A[1:4, 0]	slicing
A[1:, 1:3] = 9	assigning to parts of the matrix Slicing in python creates copy In numpy it doesn't!
np.full((2, 3), x)	2by3 matrix filled with x
view = A[0:3,2:4]	writing into a view changes underlying data
view1 = A[0:3,-2:4].copy	writing doesn't change data
np.where(-A<0, 0, A)	condition, then, else
np.c_[A[:, 0:1], A[:,2:6]]	stack slices horizontally

strings

s = input("enter here")	prompts input
s = 'x'; s = "x"	are both strings
s[0]	returns character at first position
s[0:5]	slicing
s = 'I\'m fine'	escape
s = """long string"""	enables linebreak

strings (cont)

print(s)	
\n	linebreak
+	links strings (creates a copy)
s = '{0}, beautiful {1}!'.format('Hello','World')	
s.find('x')	returns postion of first occurance
s.replace-('l','p')	replace all
s.split	returns list of words
len('string')	length
s[i]	retrieves character at position i
s[-1]	retrieves last character
s[i:j]	range i:j

lists

Same functions as strings

l = [index0, index1]	create list
l[i]	retrieves index i
l[i] = x	stores x to index i
l.insert(index,'st-ring')	doesn't remove
l.append	add to end
l.sort	smallest to largest
l.reverse	
l.remove('string') *or del l[i]*	removes first occurance
b = a.copy()	change in a doesn't effect b
b = a	change in a effects b

tuples

t = ('super', 'man')	create tuple
len(t)	returns number of items
t.count(3)	returns number of 3s
x,y = t	tuple unpacking x = t[0]; y = t[1]

tuples are immutable lists
tuples can't have one number, (1) is a math operation

functions

def my_fct(*var_a*, *var_b*):	defines function
return(*result*)	outputs result for further use

No typechecking in functions (ducktyping)
Keep scope in mind (*var_a* and *var_b* are local variables and are destroyed after functions runs
If local variable has same name as global variable, the local one will be referred to when name is used
Pitfall: `print = 5` -> predefined functions can be overwriten

lambda & mapping

lambda a, b: a + b	small nameless function that can be applied to lists etc. e.g. `sorted (list, key=lambda x: x[1])`
map	e.g. `list(m ap( lambda x: x + 10, values))`
filter	for filtering lists, e.g.: `list(f ilt er( lambda x: x % 2 == 0, value s))`

operations for Vectors & matrices

np.dot(v, w)	dot product
np.linalg.no-rm(v)	length of vector
is3 = (A == 3) A > 0	returns boolean matrix
A[A < 0] = 0	replaces entries where conditions is met (true)
np.linalg.so-lve(A, b)	solves system of linear equations
np.mean(v) A.mean(0)	0 for col max, 1 for row max
np.maximum(v, w)	
v.std()	standard deviation
v.sort()	
np.vstack(A, B) np.hstack(A, B)	vertically/horizontaly stack arrays

numpy analytics

np.minimum.accumul-ate(array)	minimum up until an entry
np.argmax(array)	index of max

math basics

a = 2 + 3j	complex num
from math import pi	π
from math import exp	e^x
np.abs(v)	
np.sqrt(v)	
np.exp(v)	
np.log2(v)	
np.sin(v)	

Cheatography

Python Basics Cheat Sheet

by LasseBoe (Lasse1618) via cheatography.com/110079/cs/21547/

dictionary

d = {'x' : 'y', 'a' : 'b'}	key:value
d['x']	returns 'y'
'y' in d	returns bool
d.keys()	returns ['x', 'a']
d.values()	returns ['y', 'b']
d.items()	returns [('x':'y'), ('a':'b')]
d['tea'] = 'Tee'	add item
d[('super', 'man')] = 'Supermann'	tuples as immutable lists
del d[k]	

key can only occur once, values can occur multiple times
(keys need to map to one value)
key: must be immutable (int, float, str, tuple (not list))
value: any data type

sets

s = {1, 2, 3}	create set
s | s2 *or s.union(s2)*	all elements of both sets
s ^ s2 *or s.symmetric_difference(s2)*	elements that are in either s or s2
s & s2 *or s.intersection(s2)*	common elements
s <= s2 *or s.issubset(s2)*	test if all elements of s is in s2 (bool)
s - s2 *or s.difference(s2)*	elements in s but not in s2

no order
identical items are combined to one item

logging

import logging

from logging import debug

logger = logging.getlogger()

logger.setLevel(logging.DEBUG)	activates logger
debug('Start')	returns message with some stats
logging.disable()	all loggers (incl debug) turned off
logging.disable(logging.NOTSET)	all loggers reactivated

try:
`command`
except:
print('oops')
or
except IndexError:
print('Index not found')
	dealing with exceptions

raise Exception ('*message*')	creates an exception
assert x>0, 'No neg. numbers'	returns error if assertion not met

other

np.repeat(1, 10)	array with ten 1s
np.asarray(*list*)	converts list to array
np.polyfit(*x_array*, *y_array*, *degree*)	fits a polynomial the points (x/y), minimizing the squared error

other (cont)

np.polyval(*coeff_array*, *x_eval_array*)	evaluates a polynomial with given coefficients (sortet highest degree to lowest!) at a number of x-point

reverse_a = a[::-1]

general numpy

np.random.randint(low=0, high=10, size=20)