<https://github.com/vinta/awesome-python>

# You have numbers

3 # => 3

# Math is what you would expect

1 + 1 # => 2

8 - 1 # => 7

10 \* 2 # => 20

35 / 5 # => 7.0

# Integer division rounds down for both positive and negative numbers.

5 // 3 # => 1

-5 // 3 # => -2

5.0 // 3.0 # => 1.0 # works on floats too

-5.0 // 3.0 # => -2.0

# The result of division is always a float

10.0 / 3 # => 3.3333333333333335

# Modulo operation

7 % 3 # => 1

# Exponentiation (x\*\*y, x to the yth power)

2\*\*3 # => 8

# You have numbers

3 # => 3

# Math is what you would expect

1 + 1 # => 2

8 - 1 # => 7

10 \* 2 # => 20

35 / 5 # => 7.0

# Integer division rounds down for both positive and negative numbers.

5 // 3 # => 1

-5 // 3 # => -2

5.0 // 3.0 # => 1.0 # works on floats too

-5.0 // 3.0 # => -2.0

# The result of division is always a float

10.0 / 3 # => 3.3333333333333335

# Modulo operation

7 % 3 # => 1

# Exponentiation (x\*\*y, x to the yth power)

2\*\*3 # => 8

# Lists store sequences

li = []

# You can start with a prefilled list

other\_li = [4, 5, 6]

# Add stuff to the end of a list with append

li.append(1)

# (It's a closed/open range for you mathy types.)

li[1:3] # => [2, 4]

# Remove arbitrary elements from a list with "del"

del li[2] # li is now [1, 2, 3]

# Get all keys as an iterable with "keys()". We need to wrap the call in list()

# to turn it into a list. We'll talk about those later. Note - for Python

# versions <3.7, dictionary key ordering is not guaranteed. Your results might

# not match the example below exactly. However, as of Python 3.7, dictionary

# items maintain the order at which they are inserted into the dictionary.

list(filled\_dict.keys()) # => ["three", "two", "one"] in Python <3.7

list(filled\_dict.keys()) # => ["one", "two", "three"] in Python 3.7+

# Get all values as an iterable with "values()". Once again we need to wrap it

# in list() to get it out of the iterable. Note - Same as above regarding key

# ordering.

list(filled\_dict.values()) # => [3, 2, 1] in Python <3.7

list(filled\_dict.values()) # => [1, 2, 3] in Python 3.7+

# Do set intersection with &

other\_set = {3, 4, 5, 6}

filled\_set & other\_set # => {3, 4, 5}

# Do set union with |

filled\_set | other\_set # => {1, 2, 3, 4, 5, 6}

# Instead of try/finally to cleanup resources you can use a with statement

with open("myfile.txt") as f:

for line in f:

print(line)

# Only evens and not after 237

numbers = [

951, 402, 984, 651, 360, 69, 408, 319, 601, 485, 980, 507, 725, 547, 544,

615, 83, 165, 141, 501, 263, 617, 865, 575, 219, 390, 984, 592, 236, 105, 942, 941,

386, 462, 47, 418, 907, 344, 236, 375, 823, 566, 597, 978, 328, 615, 953, 345,

399, 162, 758, 219, 918, 237, 412, 566, 826, 248, 866, 950, 626, 949, 687, 217,

815, 67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 379, 843, 831, 445, 742, 717,

958, 609, 842, 451, 688, 753, 854, 685, 93, 857, 440, 380, 126, 721, 328, 753, 470,

743, 527

]

for number in numbers:

if number == 237:

break

if number % 2 == 0:

print(number)

# Write a function that takes in a numerical value, and returns

# the word corresponding to that number.

# The program will handle numbers: 0 - 4, for other numbers it will

# return that the input is incorrect.

def get\_word(number):

result = ""

if number == 0:

result = "zero"

elif number == 1:

result = "one"

elif number == 2:

result = "two"

elif number == 3:

result = "three"

else:

return "input is incorrect"

return result

# Write a program that receives a list of strings and it will return the amount of variables in that list.

def amount\_of\_strings(mylist):

result = 0

for i in mylist:

if i is not str:

break

result += 1

# Write a function that receives a dictionary and it will validate if the dictionary is in the following format:

# {‘name’: String, ‘age’: Number, ‘Hobbies’: List}

def dict\_validator(dict\_to\_validate):

if dict\_to\_validate.get('name') is None or not type(dict\_to\_validate.get('name')) is str:

print("missing or incorrect type field: name")

if dict\_to\_validate.get('age') is None or not type(dict\_to\_validate.get('age')) is int:

print("missing or incorrect type field: age")

if dict\_to\_validate.get('hobbies') is None or not type(dict\_to\_validate.get('hobbies')) is list:

print("missing or incorrect type field: hobbies")