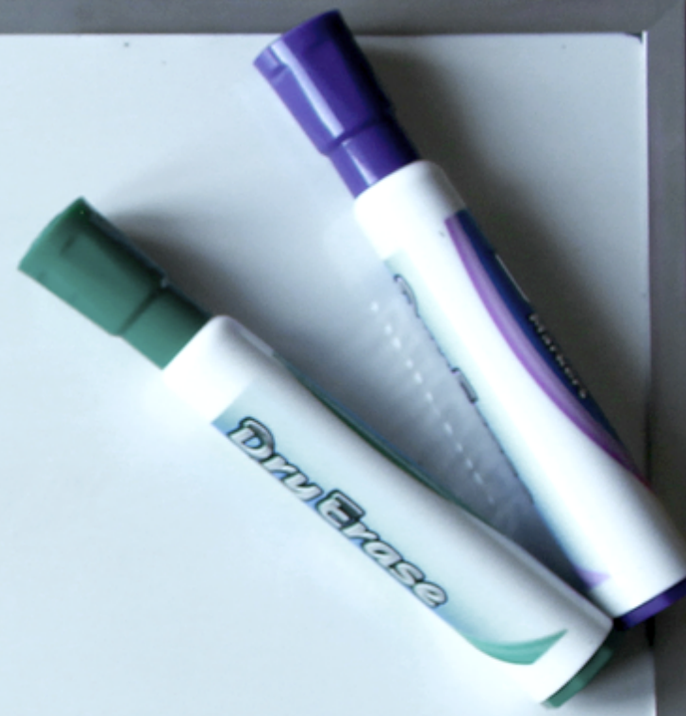
north austin pythonistas

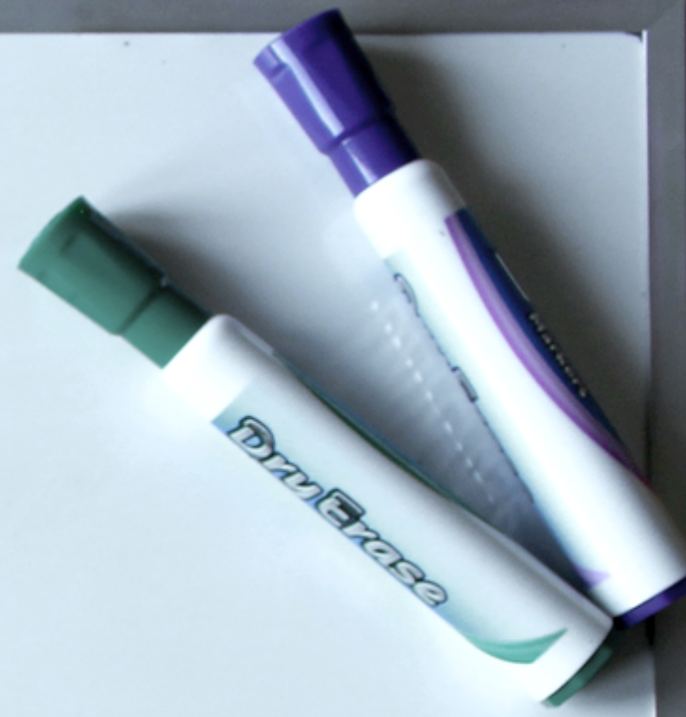
present



A Python Crash Course

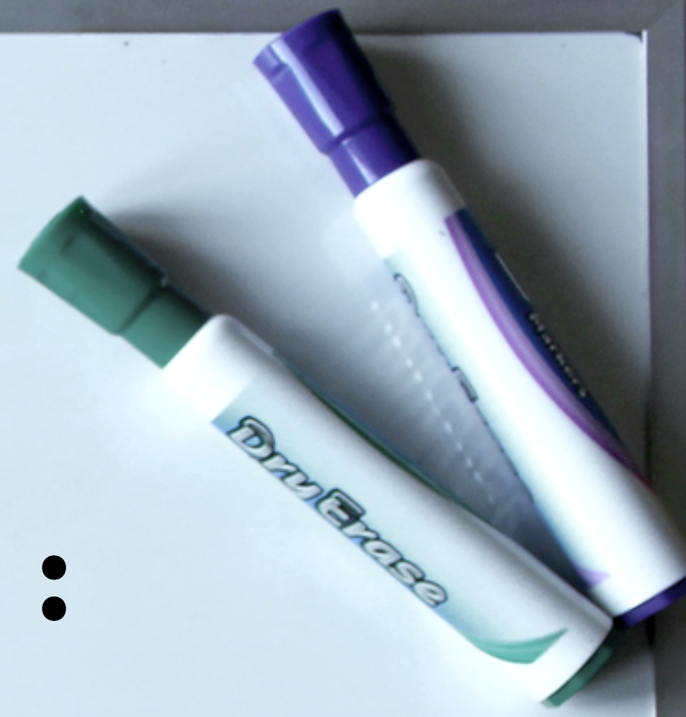
erik.oshaughnessy@gmail.com

Rules of Engagement



- Start PyCharm
- Open the “Python Console” Tab
- Type in examples as I talk
- Ask Questions!

Getting Help

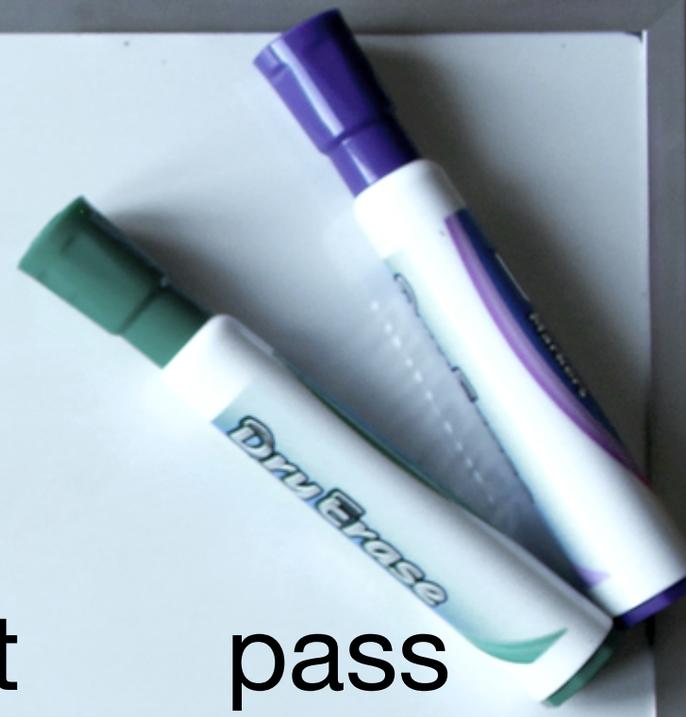


In the console (or REPL*):

```
>>> help()
>>> help(dict)
>>> help(dict.setdefault)

>>> help()
>help quit
>>>
```

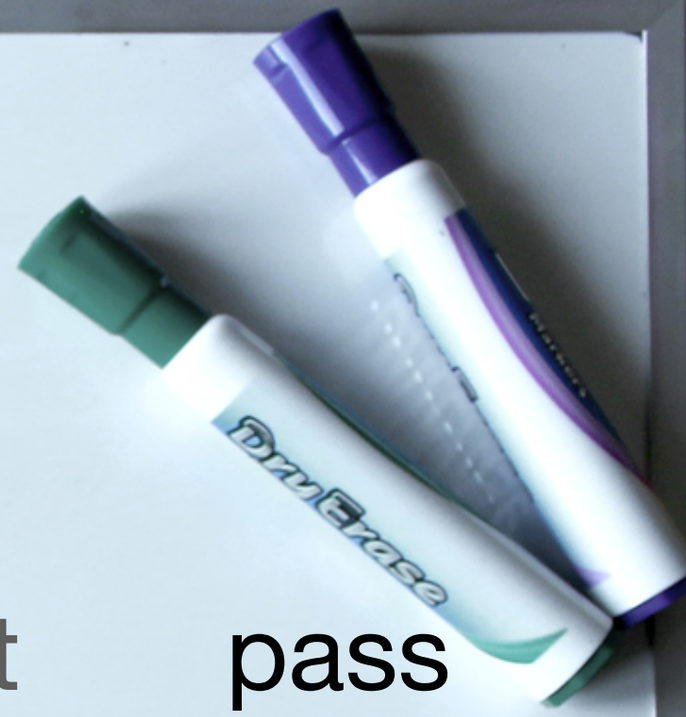
*Read Execute Print Loop

Python Keywords



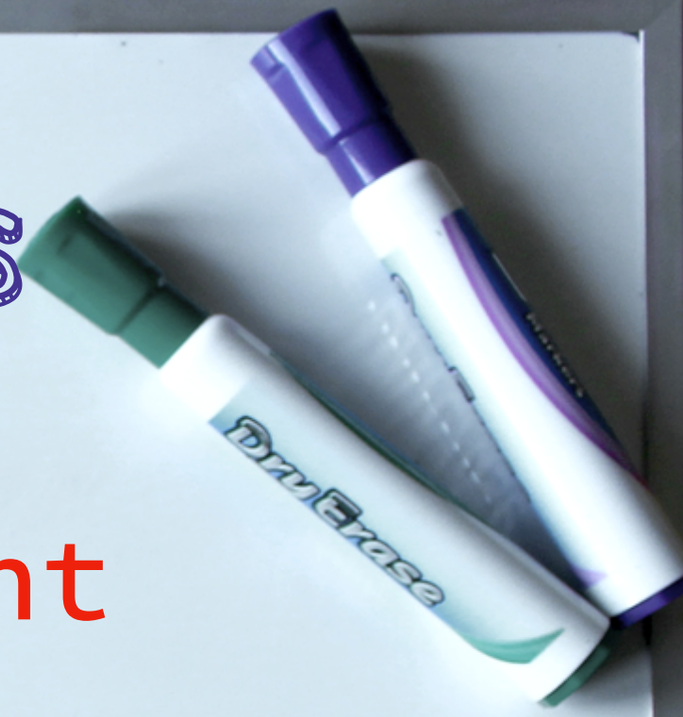
False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Python Keywords



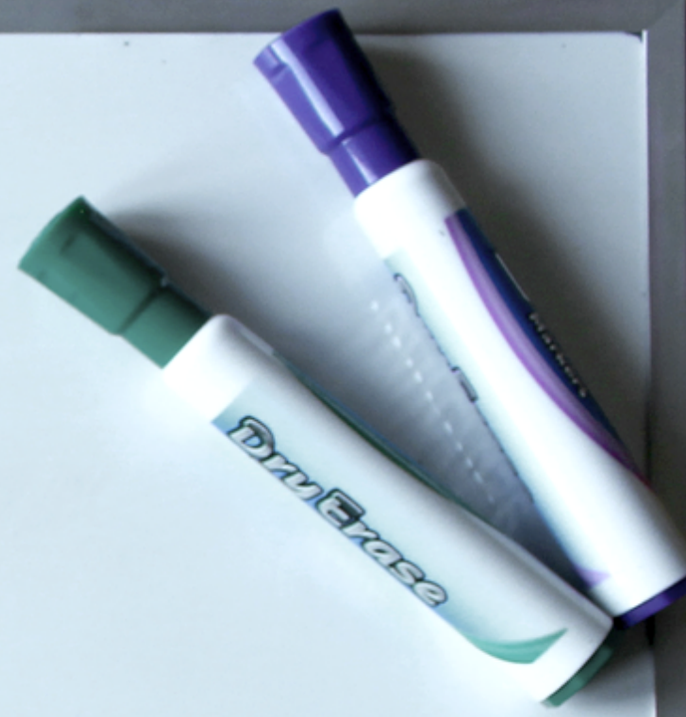
False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Simple Data Types




```
i = 1                       # int

f = 1.5                     # float

s = “letters”               # string

l = [i, f, s]               # list

d = {‘key’:’value’}         # dictionary

b = True and False          # boolean
```

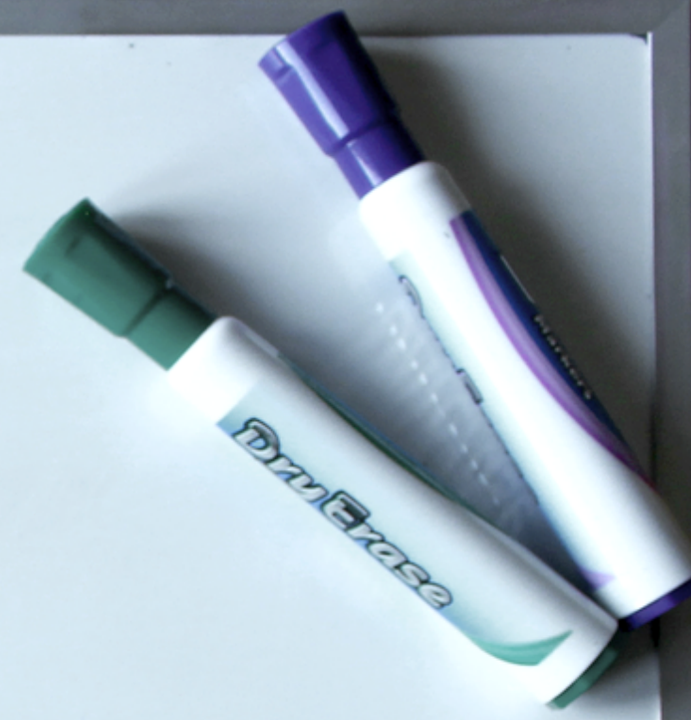
None



From docs.python.org/3:

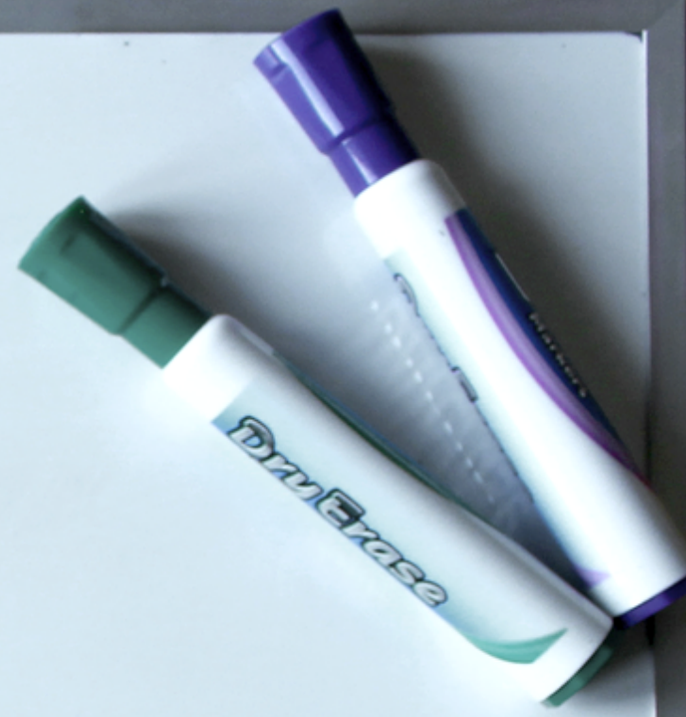
“This type has a single value. There is a single object with this value. This object is accessed through the built-in name `None`. It is used to signify the absence of a value in many situations, e.g., it is returned from functions that don’t explicitly return anything. Its truth value is false.”

None




```
foo = None

foo == None          # True

foo is None          # True

foo != None          # False

bool(foo) == False # True
```

Arithmetic Operators



`+` addition

`-` subtraction

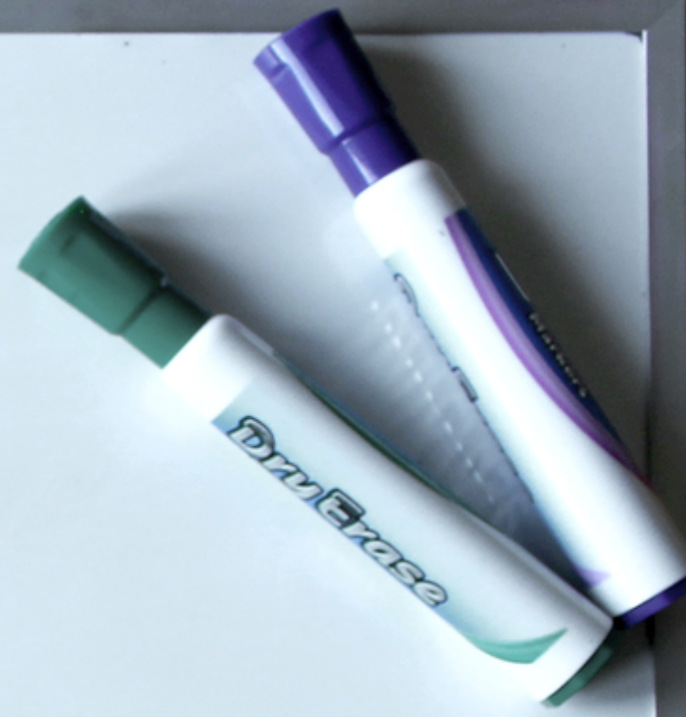
`*` multiplication

`/` true division, returns float

`//` floor division, returns

`%` modulus, returns integer

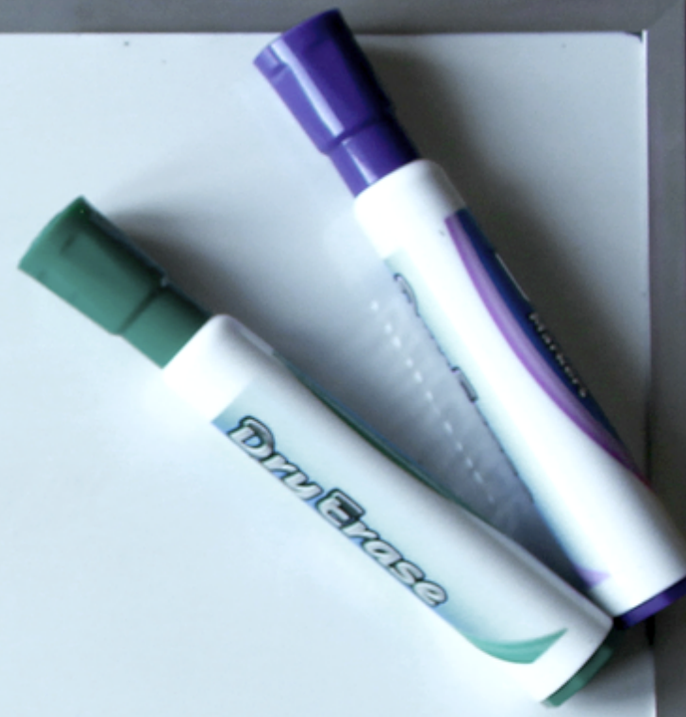
Boolean Operators



and, or, not

```
if cheese and wine and not spam:
    fancy_party()

if beer or hotdogs:
    cookout()
```

Comparison Operators



`>` greater than

`>=` greater than or equal

`<` less than

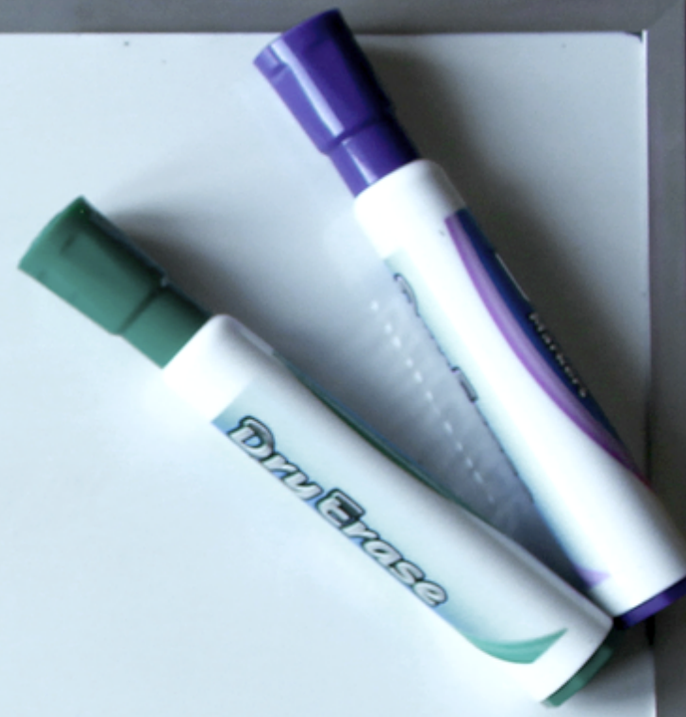
`<=` less than or equal

`==` equal

`!=` not equal

All comparisons result in boolean values

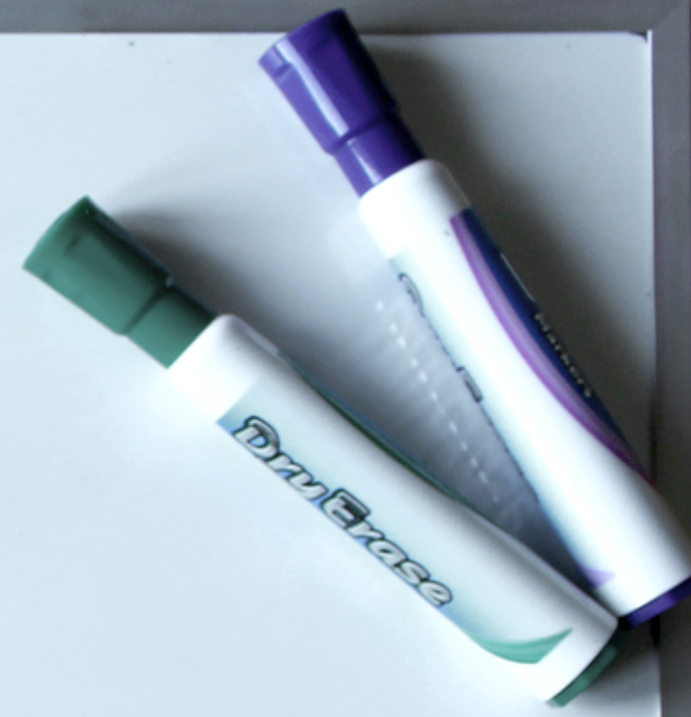
Flow Control
if




```
if boolean_statement:
    do_true_activities()
```

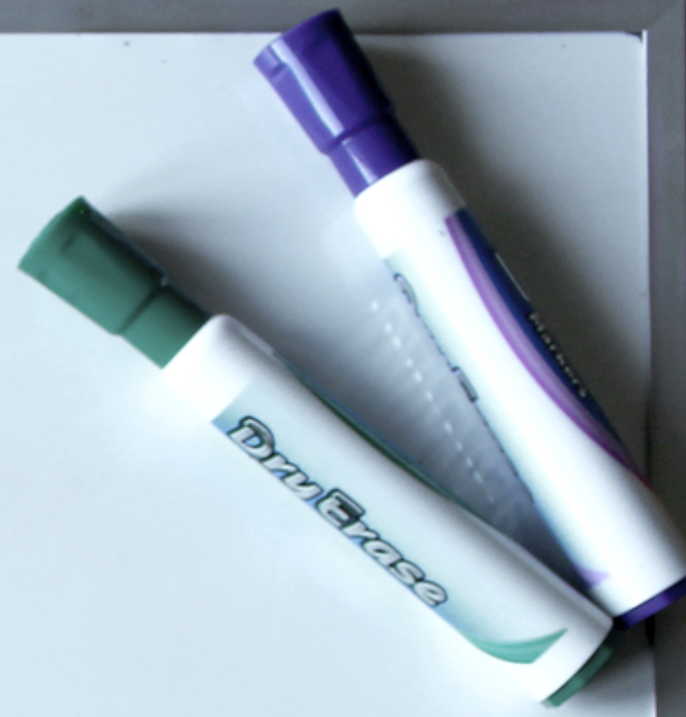
```
if number_of_trombones > 11:
    print(“plenty of trombones”)
```

Flow Control
if-else



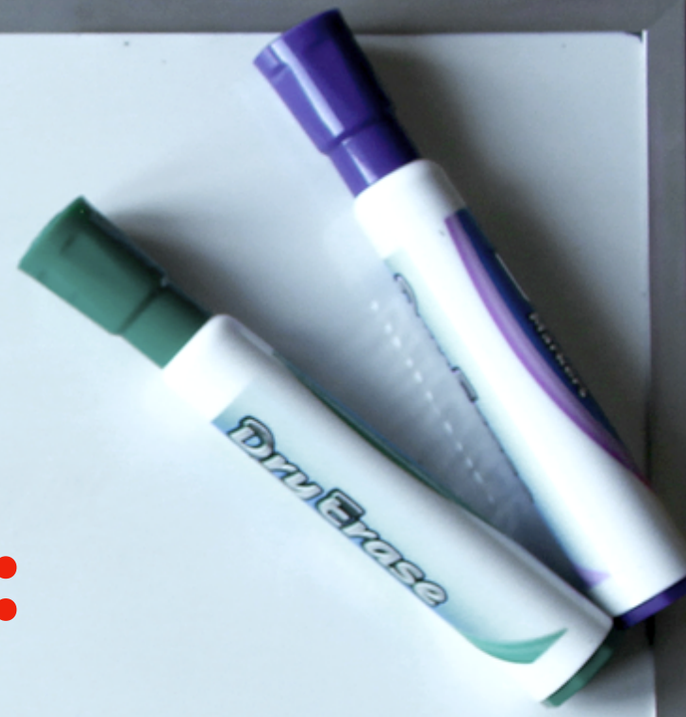

```
if boolean_statement:
    do_true_activities()
else:
    do_false_activities()
```

Flow Control
if-elif-else




```
if boolean_statement:
    do_true_activities()
elif another_boolean_statement:
    do_other_true_things()
else:
    do_false_activities()
```

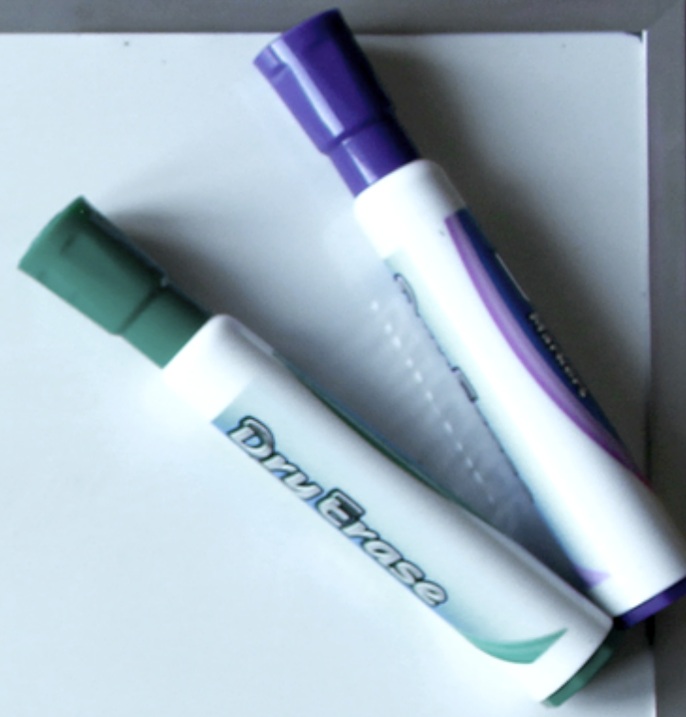
Flow Control
for




```
for variable in iterable:
    statements
```

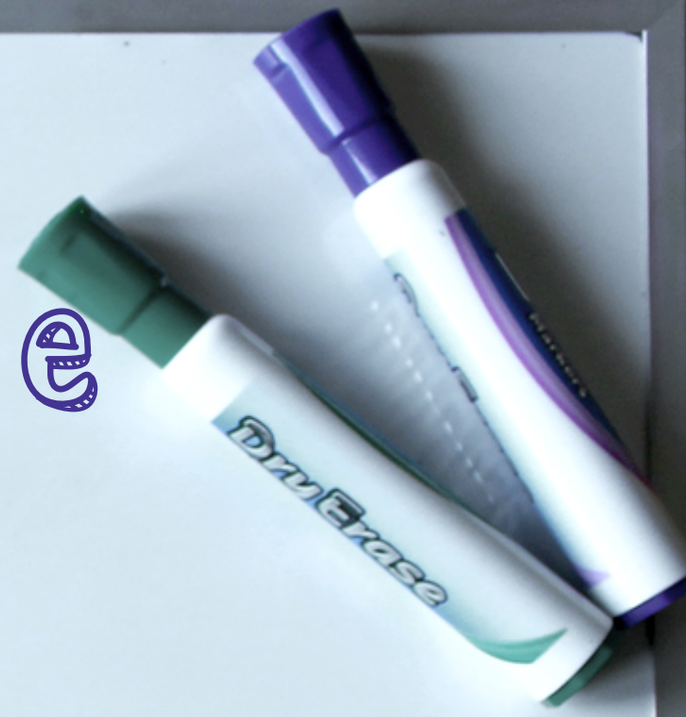
```
for n in [‘mary’, ‘sue’, ‘jane’]:
    print(n)
```

Flow Control
For with a break



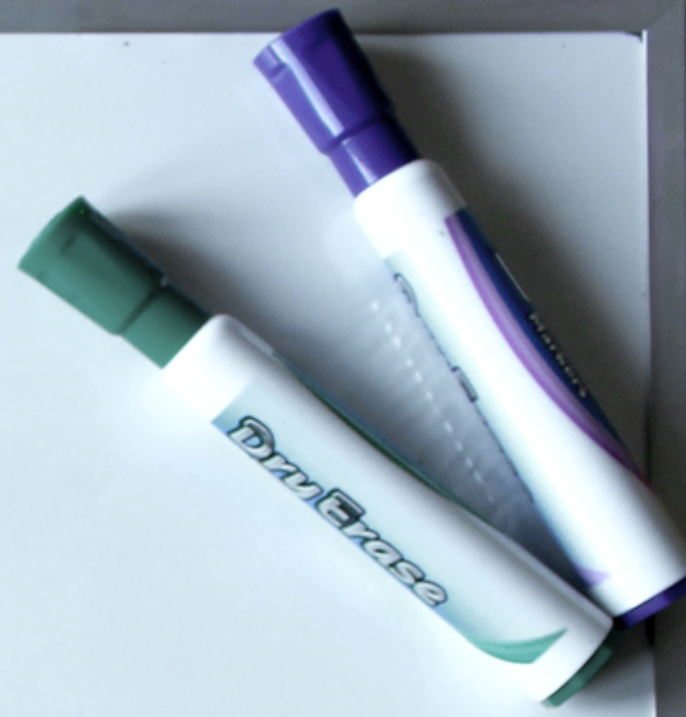

```
for n in [‘mary’, ‘sue’, ‘jane’]:
    if n == ‘sue’:
        break
    print(n)
```

Flow Control
For with a continue




```
for n in [‘mary’, ‘sue’, ‘jane’]:
    if n == ‘sue’:
        continue
    print(n)
```

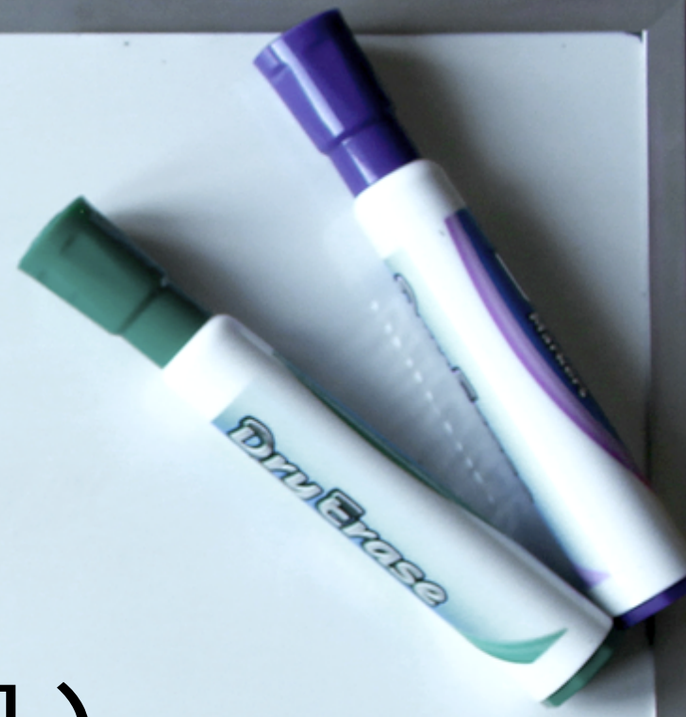
Flow Control
for with a range




```
for n in range(0, 3):

    print(n)
```

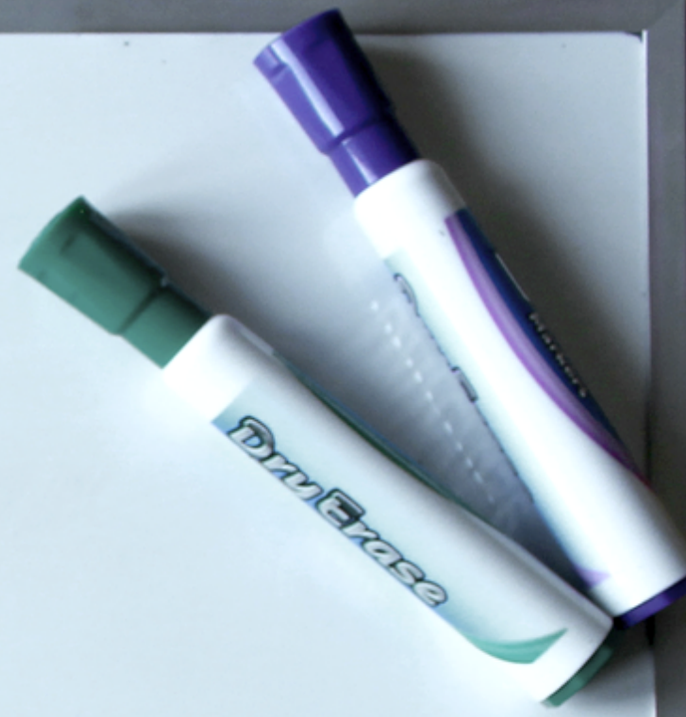
```
0

1

2
```

Flow Control
range builtin




```
range(stop)
range(start, stop, [step])

Returns a list of integers.

range(4)          => [0, 1, 2, 3]
range(0, 4)       => [0, 1, 2, 3]
range(0, 4, 2)    => [0, 2]
```

Flow Control
while

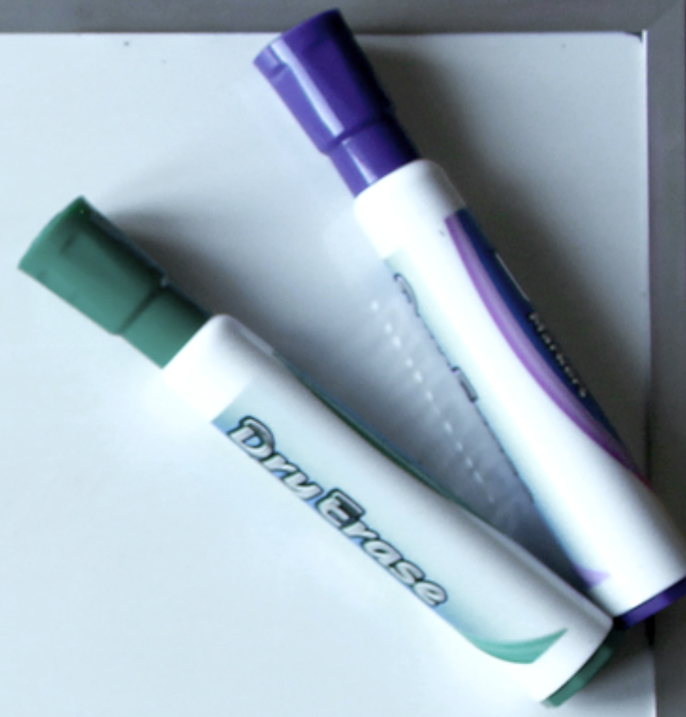



```
while condition_is_true:
        body()
```

```
while True:
        print(“hello world”)
```

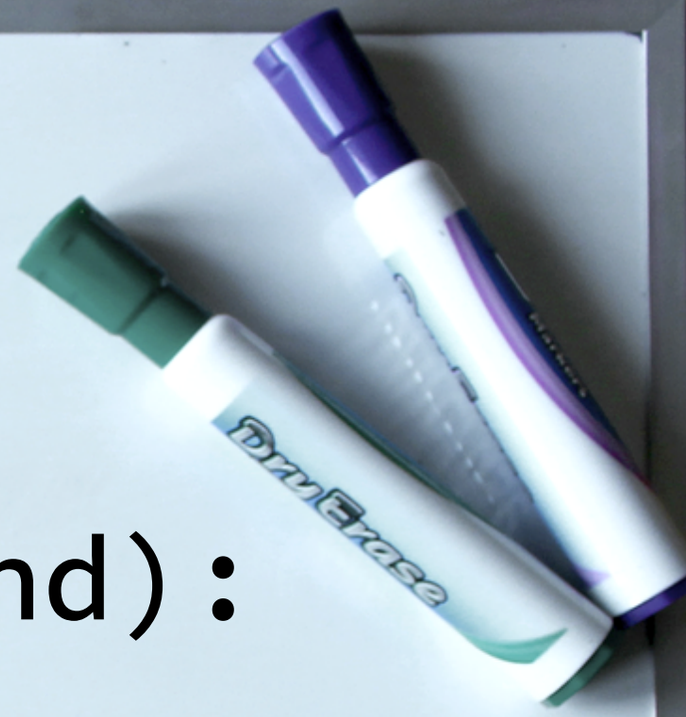
(Type control-c to stop)

Flow Control
while with a break



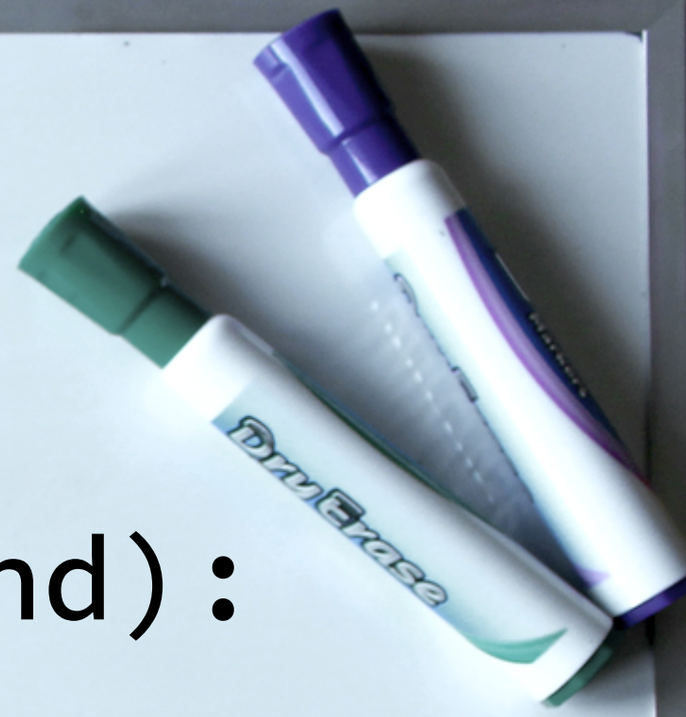

```
while True:
        name = input(“name?”)
    if name == ‘bye’:
        break
    print(‘hello’, name)
```

Functions



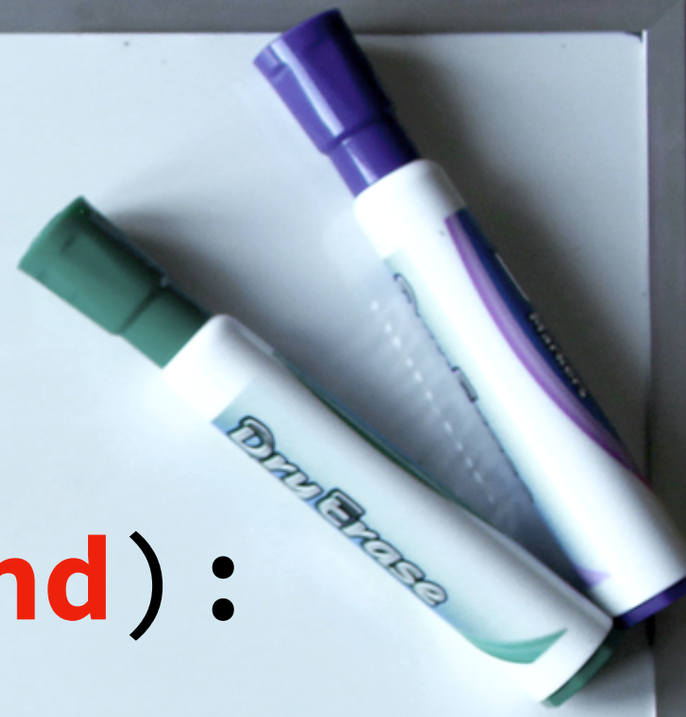

```
def packager(first, second):
    '''Package arguments.'''
    package = { 'a': first,
                'b': second }
    return package
```

Functions



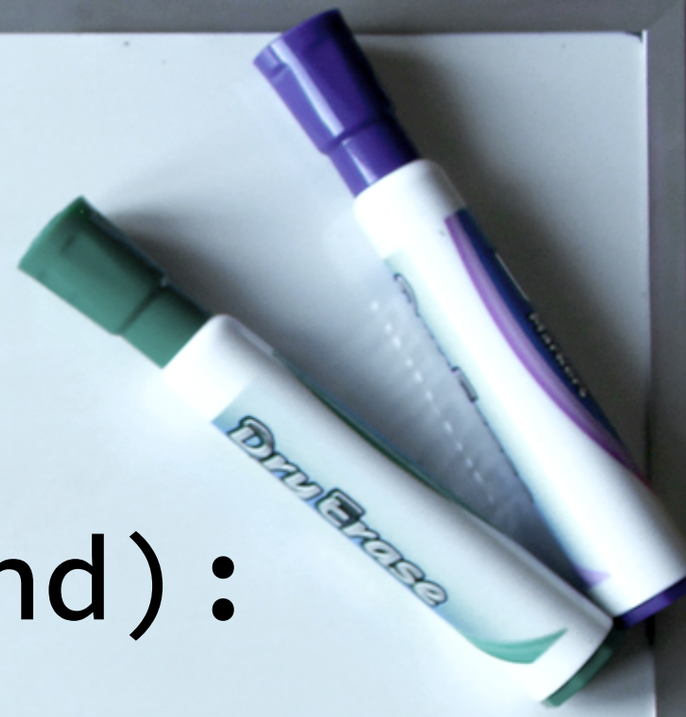

```
def packager(first, second):
    '''Package arguments.'''
    package = { 'a': first,
                'b': second }
    return package
```

Functions



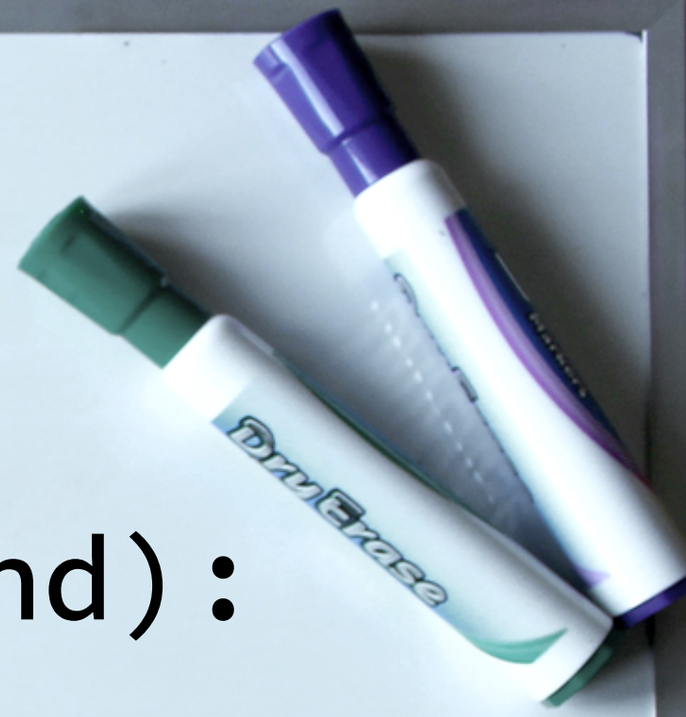

```
def packager(first, second):
    '''Package arguments.'''
    package = { 'a': first,
                'b': second }
    return package
```

Functions



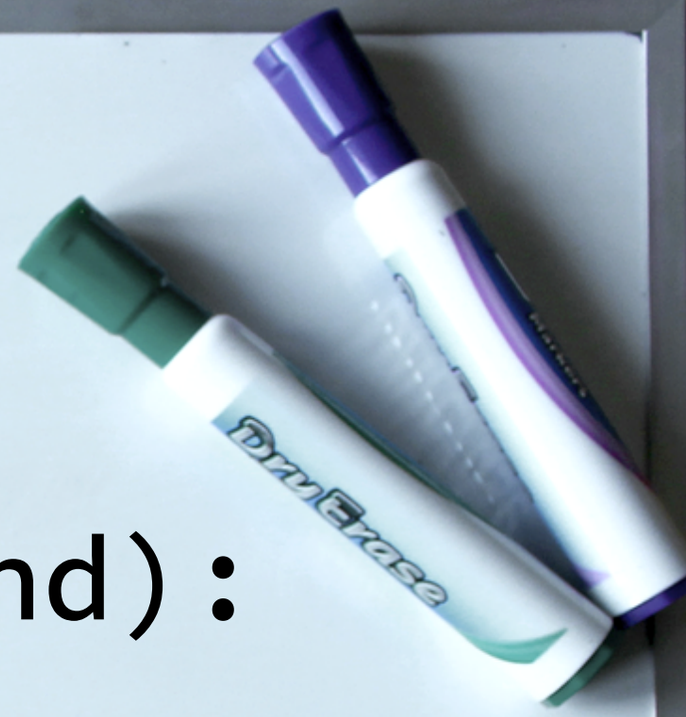

```
def packager(first, second):
    '''Package arguments.'''
    package = { 'a': first,
                'b': second }
    return package
```

Functions



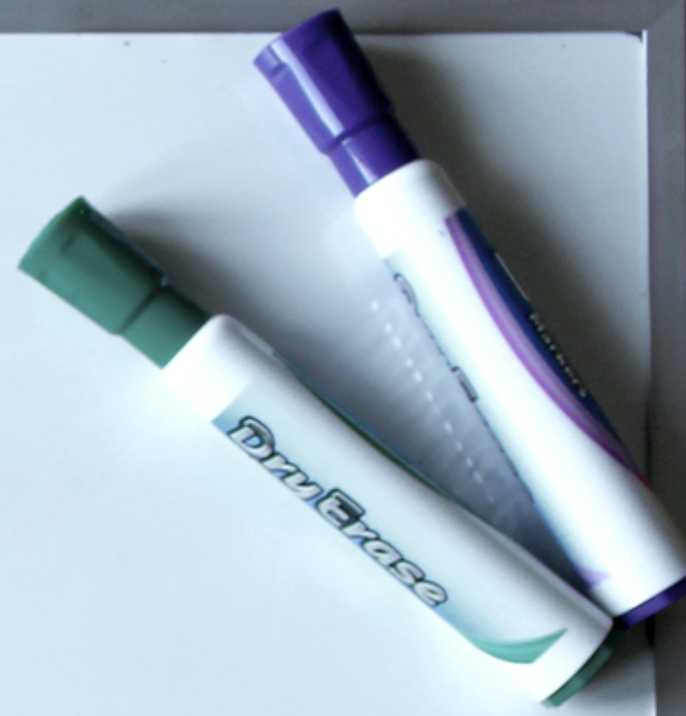

```
def packager(first, second):
    '''Package arguments.'''
    package = { 'a': first,
                'b': second }
    return package
```

Functions



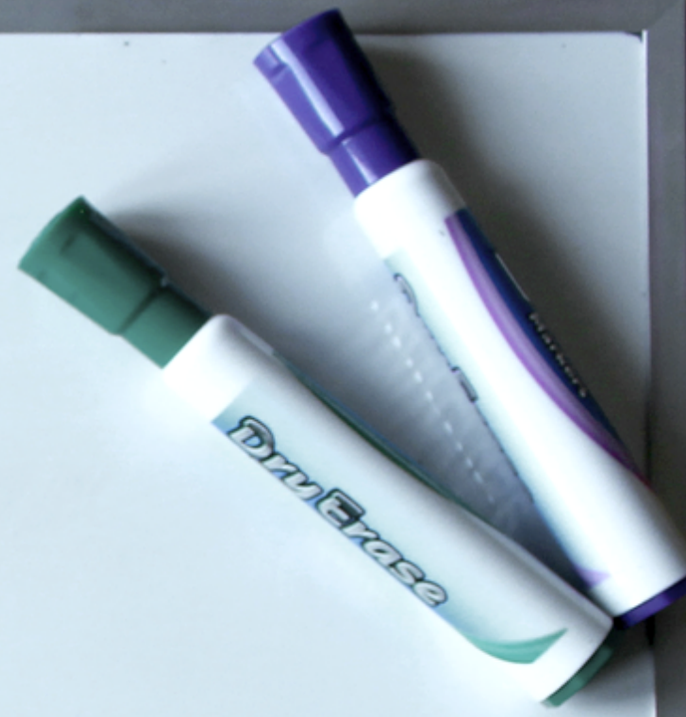

```
def packager(first, second):
    '''Package arguments.'''
    package = { 'a': first,
                'b': second }
    return package
```

Calling Functions



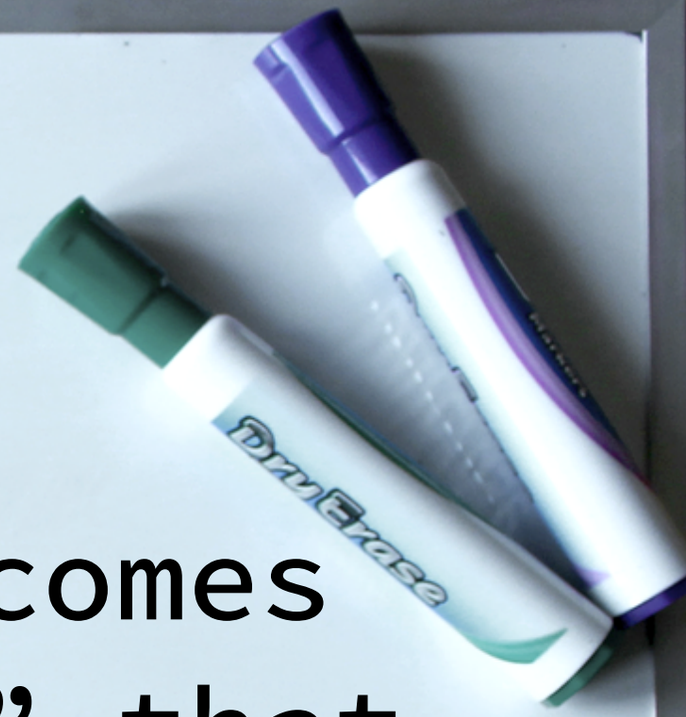

```
pkgA = packager(“tacos”,
                “dr pepper”)
pkgB = packager(“hamburger”,
                “shake”)
```

Some Useful Builtin Functions




```
min()
>>> min([3, 8, 99])

max()
>>> max([3, 8, 99])

len()
>>> len([3, 8, 99])
```

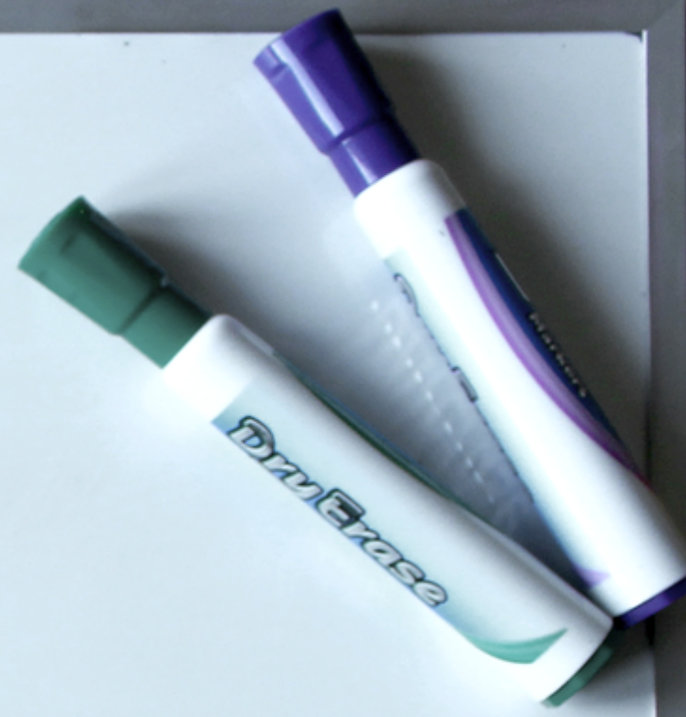
Batteries Included



The python distribution comes with a standard “library” that provides packages that solve different problems.

```
>>> help()
help> modules
help> os
```

import




```
import module
```

```
import os

if os.name == ‘posix’:
    print(os.uname())
```

from - import


```
from module import entity

from os import uname

print(uname())
print(os.name)   # error!
```

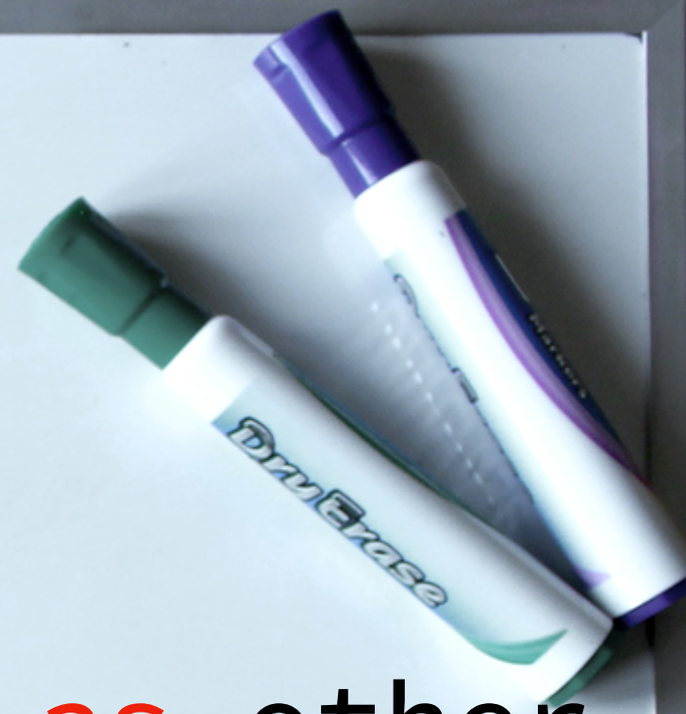
import - as


```
import module as new_name
```

```
import numpy as np
```

```
a = np.ndarray()
```

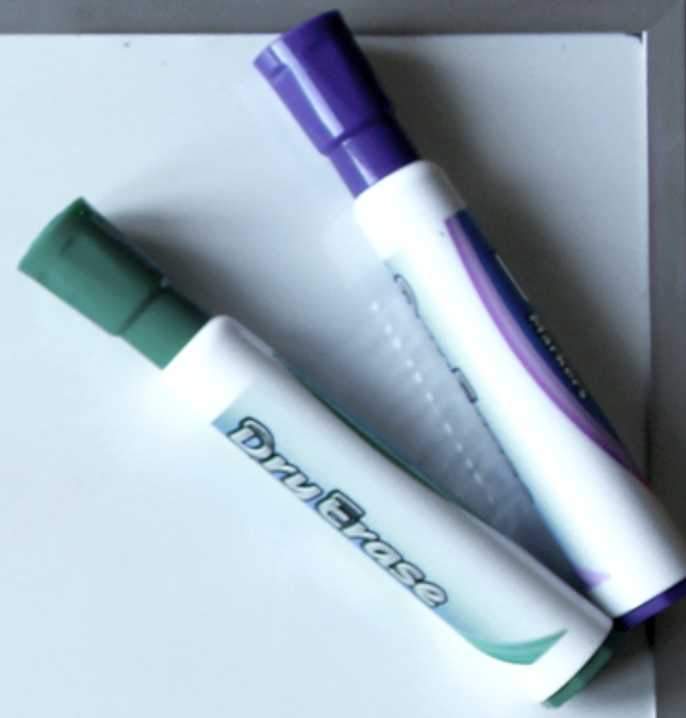
from - import - as




```
from module import entity as other
```

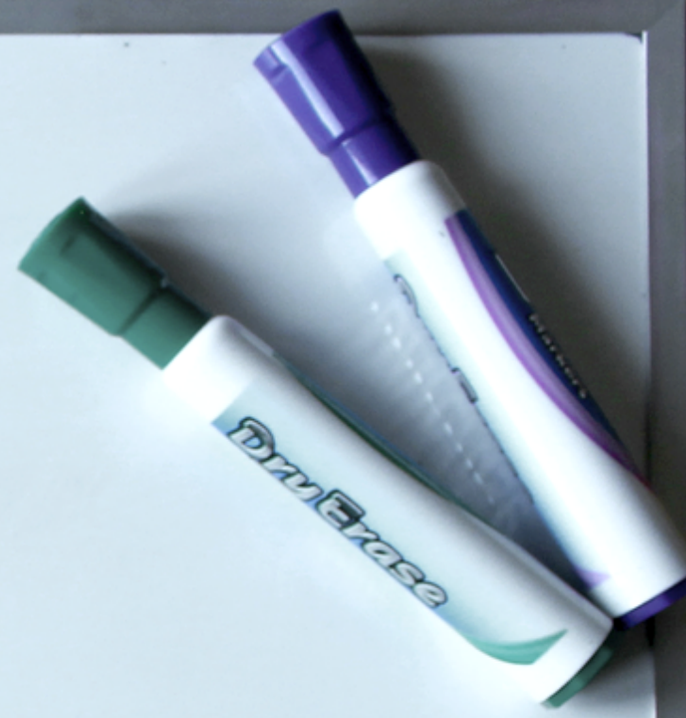
```
from os import uname as UNAME

def uname():
    results = UNAME()
    print('uname said', results)
```

try - except



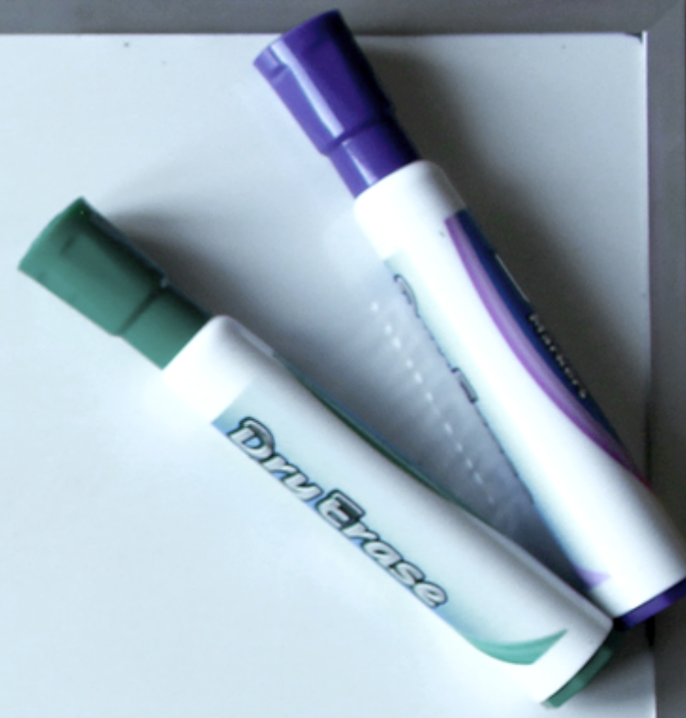

```
try:
        statements
        that_might
        raise_an_exception
except [[exception] as name]:
        handle_the_exception

>>> help()
builtins
```

try - except



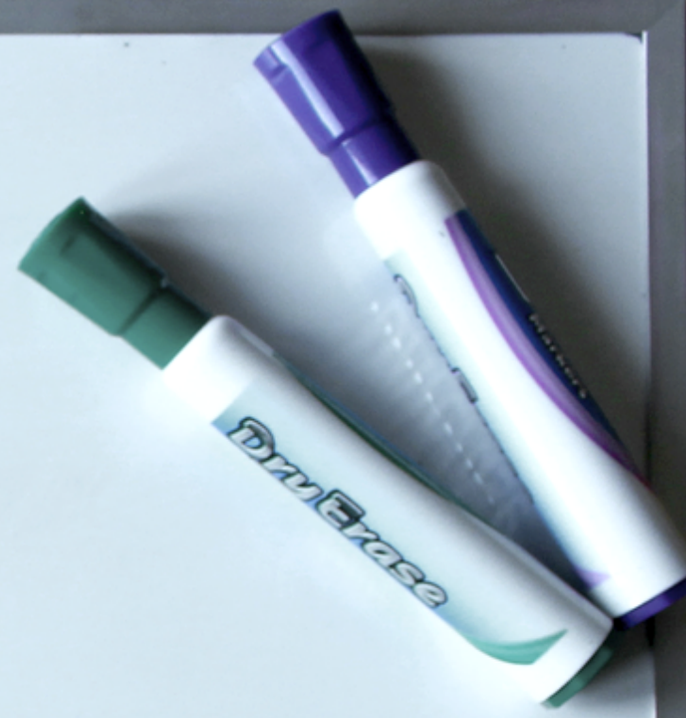

```
try:
      n = 11 / 0
except:
      print(“11 / 0 failed”)
print(n)
```

try - except



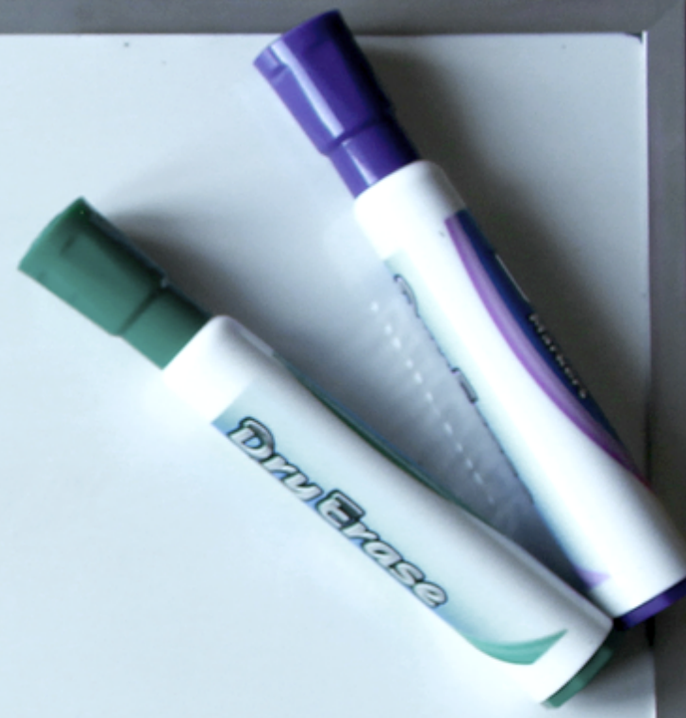

```
try:
      n = 11 / 0
except Exception as e:
      print(“11 / 0 failed”, e)
print(n)
```

try - except



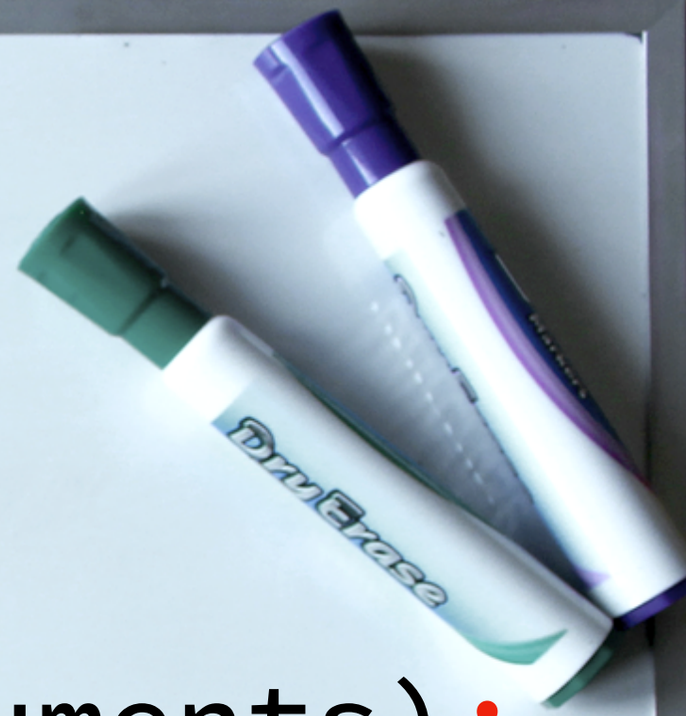

```
try:
      n = 11 / 0
except ZeroDivisionError as e:
      print(“11 / 0 failed”, e)
print(n)
```

try - except-raise



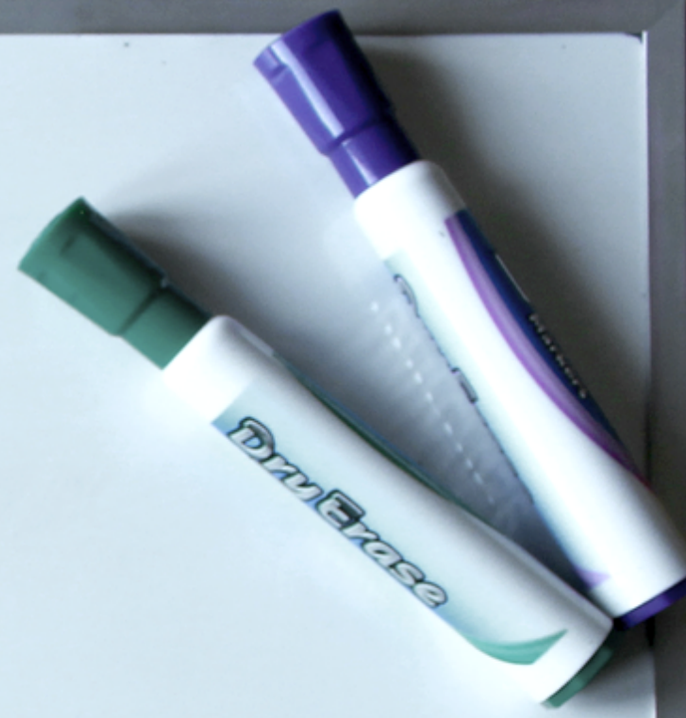

```
try:
      n = 11 / 0
except ZeroDivisionError as e:
      print(“11 / 0 failed”, e)
      raise e
print(n)
```

Python Classes



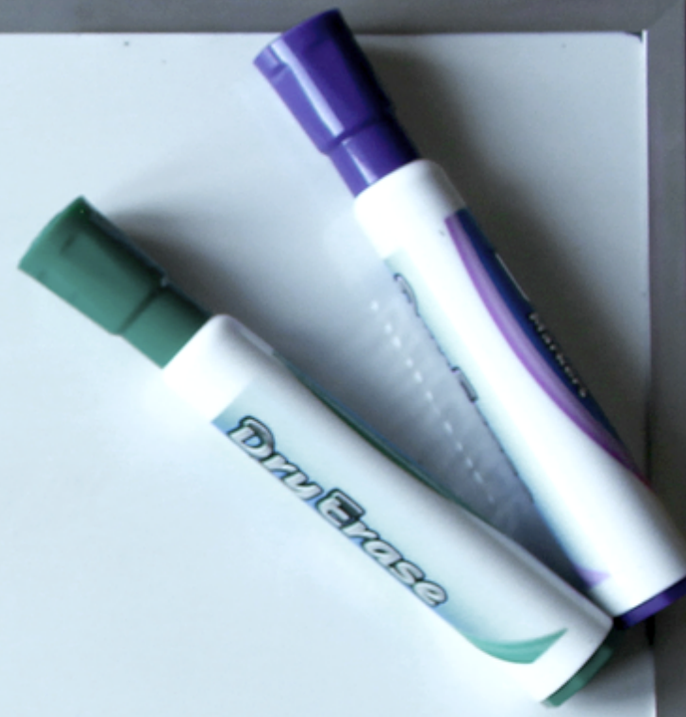

```
class Foo:
    def __init__(self, arguments):
        self.args = arguments
    def method(self, first):
        print(self.args, first)

foo = Foo(10)

foo.method(20)
```

Python Subclassing




```
class Animal:
    def __init__(self):
        self.alive = True
        self.has_a_spine = True

class Invertebrate(Animal):
    def __init__(self):
        super().__init__()
        self.has_a_spine = False
```

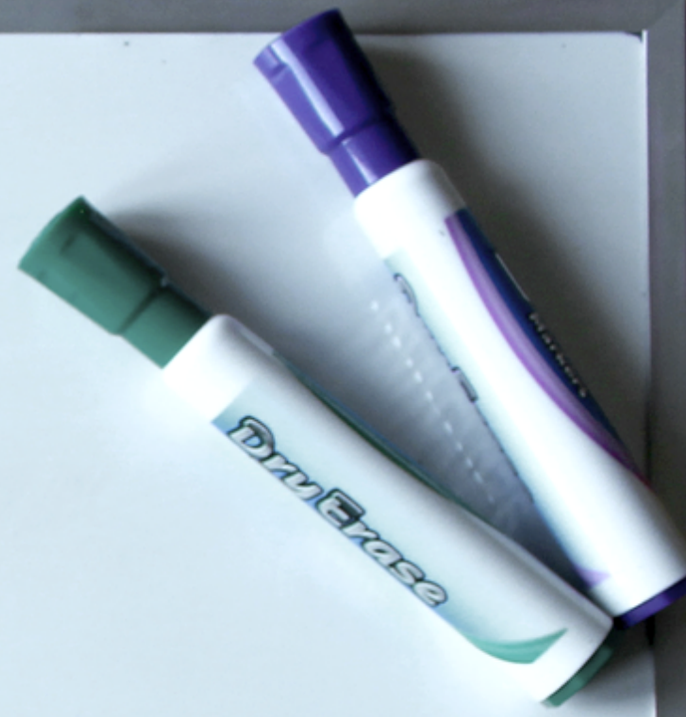
Python Subclassing




```
aardvark = Animal()
aardvark.is_alive == True
aardvark.has_a_spine == True
```

```
mollusk = Invertebrate()
mollusk.is_alive == True
mollusk.has_a_spine == False
```

Find More Help



https://docs.python.org/3

https://stackoverflow.com

https://pybit.es

Slack: pybites.slack.com

https://realpython.com

https://talkpython.fm