Lecture 2

print ("Hello" + 7) doesn't work

print ("Hello" * 7) print hello 7 times

True = 1
False = 0

print (25 + (3 < 5)) prints 26

print (print ("Hello")) prints Hello
prints the data type "none"

print (type (3)) prints <class 'int'>

print (int ("3") + int ("5")) prints 8
print (str (3) + str (5)) prints 35
print (int ("Hello")) crashes
print (int ("13.5")) crashes

Declare variable

x = 10
print (x)

Input

input ("Please type something:")

''' — triple quote with comment

— single line

Variable names (user_input)

space (not recognised).

```
user_input = input("---:")
print("  ", user_input)

print(type(print("Hello")))
```

"Hello"

None

<class 'None Type'>

None

```
grade = int(input("Give me your first grade:"))
print("...", (grade+0)/2)
```

Function

```
def function_Name():
    print ---
    return 1469

print ...
function_Name()
x = function_Name   # x holds 1469
print(x)
```

```
def function_Name(a,b,c)
    print(a,b,c)

function_Name(1,2,3)
function_Name("A","B","C")
function_Name(True,"Hello",126)
```

---

return means finish the function

Global var - • can't be changed from inside function
• declare variable over function

```
name = "Bruno"
function():
    function()
    print(name)
```

Function not able to change name
Name = Bruno

Global & Local Var

Memory Model

x = 7
y = "Hello"
z = 5
a = 7
b = "Hello
x = 9
x = y

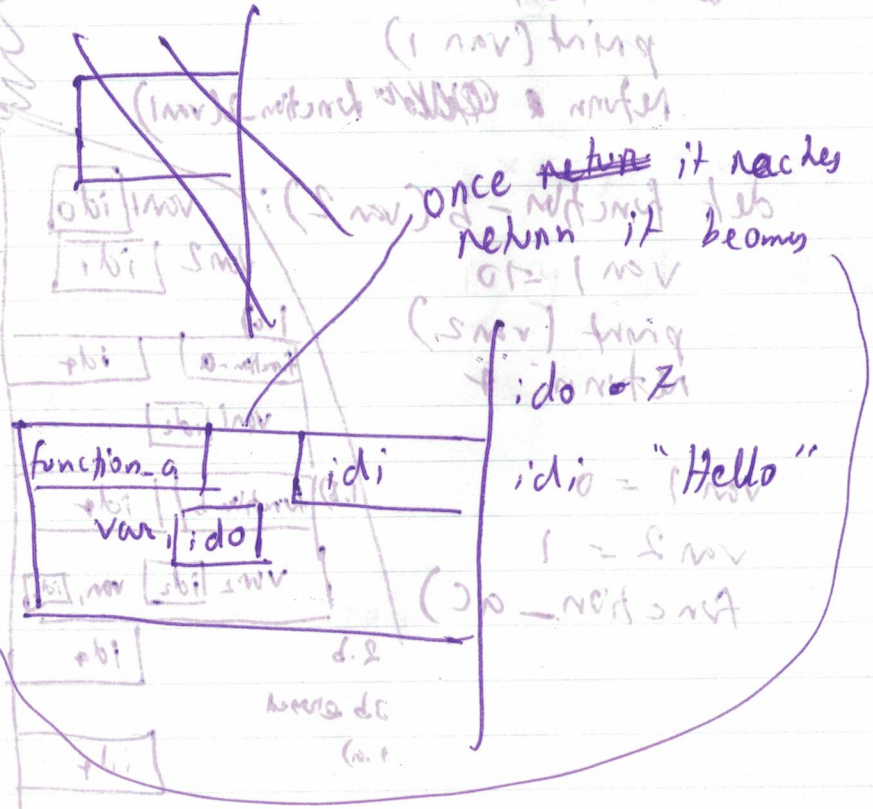
Var space

~~x id1~~ ~~x id3~~ ~~x id0~~
y id1
z id2
a id0
b id4

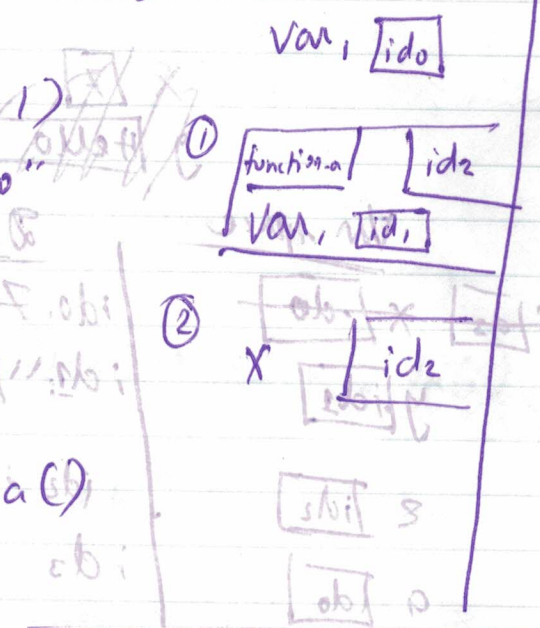
Data space

id0 : 7
id1 : "Hello"
id2 : 5
id3 : 9

```
def function_a():
    var 1 = 7
    return "Hello"

x = function_a():
```

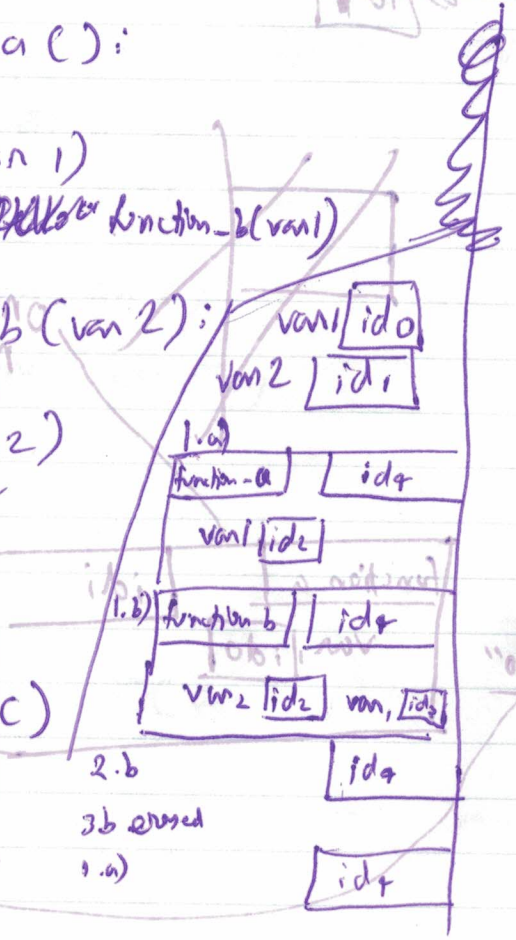


```
def function_a():
    var1 = 7
    print (var1)
    return "Hello"

var1 = 10
print(var1)
x = function_a()
print (var1)
```



id0 : 10 >>> 10
id1 : 7 7
id2 : "Hello" 10

```
def function_a():
    var1 = 7
    print (var1)
    return function_b(var1)

def function_b (var2):
    var1 = 10
    print (var2)
    return 4

var1 = 0
var2 = 1
function_a()
```



id0 Error >>> 7
id1 1 7
id2 7 0, 1, 4
id3 10
id4 4

Lecture 3

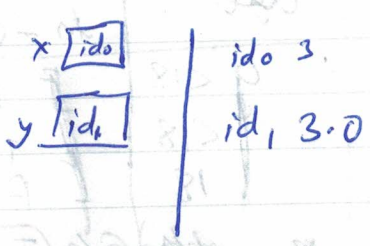
- Boolean ('bool')

- 6 common comparison operators
  - > greater than
  - < less than
  - >= greater than or equal to
  - <= ~~greater~~ less than or equal to
  - == equal to
  - != not equal to

"=" used for assignments

is - not used very often in this course

```
x = 3
y = 3.0
print (x == y)      True
print (x is y)      false
```



is x pointing to the same memory location as y

x == y 3 vs 3.0 ✓

x is y id0 vs id1 X

and	FALSE AND TRUE	FALSE
or	FALSE OR TRUE	TRUE
not	opposite	

Use brackets when lots of boolean

95 93 92 85 80 81

TRUTH TABLE

Code: result = is_tall and is_female

is_tall	is_female	result
F	F	F
F	T	F
T	F	F
T	T	T

F = 0
T = 1

Code: def can_i_vote(age, is_citizen):

age	is_citizen	return value
9	F	F
<18	T	F
<18	F	F
18	T	T
>18	F	F
>18	T	T

help("function")

int
bool
str
float

help()

Design Recipe

-8 steps

1. Header: the function definition
2. Type contract: What types come in, what type is returned
3. Requirements: Limits on the inputs other ~~that~~ than type
4. Examples: Some example inputs/outputs
5. Description: Explanation of what the function does
6. Internal Comments: Plan of what your code will do
7. Code: The body of the function
8. Test: Make sure that your function works.

Header: ~~name~~ def name (parameters) should be defined.

Type contract: ''' (float) -> float
-input -output
'''

Requirements: REQ: area > 0

Examples: I Gap
>>> get_radius(100)
5.64 ......

#Show user basics of what func does
#can provid more than 1 example
some func example not needed. Ex - rand_int eidk it works python

Description: ~~Reden~~ After Type contract
Description of what code does. Someone who does not look at the internal code.
Ex - Return the radius of a circle with area = area

Internal comments: # first check if the user is age appropriate
then return statement.

Main Logic → Explain basic steps of what the program does. Couple lines, explaining basic steps of func.

Code: now write code between internal comments
Dont think of logic, Do exactly as internal comments tell.

```
def can_i_drink(age, country):
    """ (int, str) -> bool
    REQ: age >= 0
    REQ: country in {Canada, USA, FRANCE, Saudi}
    >>> can_i_drink(15, "Canada")
    False
    >>> can_i_drink(35, "Canada")
    True

    Return true iff a person of age years old can drink in the country denoted by country.
    PUT RULES DOWN
    """
    # create a boolean that's true for Canadian drinking
```

age	Country	result
< 19	Canada	F
≥ 19	Canada	T
< 21	USA	F
≥ 21	USA	T
< 18	France	F
≥ 18	France	T
Any	Saudi	F

if & only if

write code with comments

vs French

Mid term
BV Gym / R Gym Till Week 3 material Know Tutorial #

Lecture 4

len() - space counts
.upper() - Ex- print(my_string.upper())

print(my_string[0]) - "H" (starts from zero)
print(my_string[2:9]) pieces of string

LISTS - new data type
- sequential collection of values denoted by []

```
my_list = [1, 2, 3]
print(len(my_list))
```

A lot of string and lists same.

```
print(my_list[2])
my_list[2] = 99
my_list = [1, 2, 3, "A", "B", "C", True, False, 12.3]
```

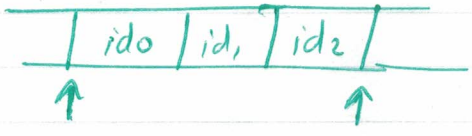
~~can do~~

Can have lists in lists

```
print(my_list[6][2][1])
```

my_list [id3]

id0 1
id1 2
id2 3
id3 [id0, id1, id2]



```
if condition:
    block of code
else:
    code
elif condition:
    block of code
else:
    code
```

Lecture 5

```
while (condition):
    block
```

Indexed for loop

```
for counter in range(start, end, skip)
    block
```

0, 11, 1

0 10

* if skip value not given go up by 1
* if start value not given start from 0

Elemental for loop

```
my-string = "hello world"
for next_letter in my-string:
    print (next_letter)
```

5 %(mod) 2 = 1 odd
4 % 2 = 0 even

```
printed_last_time = False
if (printed_last_time)
    printed_last_time = False
else
    printed_last_time = True
```

Lecture 6

WordPad. Save as plain text document. Same folder as code.

Open a file

Open(filename, mode)

- (str, str) → io.TextIOWrapper → File Handle
- mode can take several values:
  - r: open file for reading
  - w: open file for writing (erase whatever in it)
  - a: open file for adding more to end of file

Close file

- filehandle.close()
  - (File Handle) → NoneType

File handle ↓

my_file = open("test_file.txt", "r") ← File name

never leave code open →

block of code

my_file.close() ←

Read from file

filehandle.readline() → reads single line from file

- reads the first line, then next and so on

```
ex. first_line = filehandle.readline()
    second_line = filehandle.readline()
```

filehandle.read() → reads entire document

ex. all_text = my_file.read()

~~file.readlines~~

filename.readlines() → whole file into list

```
all_text = my_file.readlines()
```

save to memory

```
my_file = open("test_file.txt", "r")
all_text = my_file.read()
my_file.close()

print(all_text)
```

Good form

Open & close file to save to memory and print

```
open file
all_text = my_file.readlines()
close file.
for next_line in all_text:
    next_line = next_line.strip()
    print(next_line)
```

```
my_file = open
    for next_line = next_line.strip()
    print(next_line)
my_file.close()
```

not holding much in memory but keeps file open for longer

~~var[2:9]~~

Write to file

~~file=open~~ my_file = open("...", "a")

my_file.write("...") ← can only write strings

Adds in one line ∴ ~~add~~
("...\n") this adds more line

Functions cannot change immutable global var

int, string, booleans are immutable.
lists are mutable

only parts of list can be changed by func. The entire list cannot be changed by func. Whenever list mutated always tell reader.

ex = func' can change my_list[0] = "x" ← mutation

but cannot change my_list = ["a", "b"] ← no mutation

do not edit list in function without making it very clear. Rule of thumb: don't mutate in func if not asked.

Cloning Aliasing

```
x = ["A", "B", "C"]
y = x
x[0] = "MUTATED"
print (x,y)
```

>>> ['MUTATED', 'B', 'C']
['MUTATED', 'B', 'C']

y is another name for x. Any of them change then the other changes. Same spot in memory

Clone

```
x = ["A", "B", "C"]
y = x[:]
```

assumes beginning ← assumes end

```
x[0] = "MUTATED"
print (x,y)
```

>>> ['MUTATED', 'B', 'C']
['A', 'B', 'C']

x & y points to different position

CLONES ONLY CREATE SHALLOW COPIES

If list inside list clone list as well

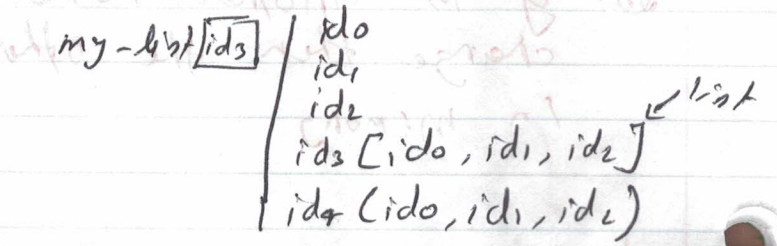
Term test to mutability

Lecture 7

- Tuple - immutable, same properties as list

```
my_list = [1, 2, 3]
my_tuple = (1, 2, 3)

print (my_list)
print (my_tuple)
```

* lot of the same things that work for list work for tuples



- features of tuples. Easy to do large number of assignment

(name, address, phone_number, office) = ("Brian", "123 Main St", "519-...", "SW")

```
x=10
y=2
print (x,y)
```

Swap values of x and y

```
temp = x
x = y
y = temp
```

(x, y) = (y, x)

- tupples can return more than one value. Can do it with lists as well but common practice is to do it with tuples

my_set = {1, 2, 3}

- there is no order or repitition

{1, 2, 3} is equivalent to {2, 1, 3}
= {1, 3, 2, 2, 3, 3, 1, 3, 2}

C - subset ∩ - intersect ∪ - union

my_set = {3, 2, 1}

print (my_set)

1, 2, 3

my_set = {3, 2, 1, 1, 1, 3, 2}

print (my_set)

1, 2, 3

my_set = {3, 2, 1, 2, 3, 1, "Hello", 9.3, "Goodbye", (1, 2, 3, 3.14)}

print (my_set)

Random order

Sets have random order

---

my_set = {2+1, 2, 1, 1, 2, 3, 1, 2, 1, 3, 1}

~~print my_set~~

my_set.add(2)

my_set.add(4)

print (my_set)

my_set | ids
id0 3
id1 2
id2 1
id3 { id1, id0, id2, id4 } any order. order doesn't matter

for next_element in my_set:
    print (next_element)

for i in range (len(my_set)):
    print (my_set[i]) ← ERROR. For loop over set elemental for loop.

word_list = training_file.read().split()

What can I do with set that I couldn't with list?
- only unique elements in list

print (4 in list)
print (4 in my_set)

- can do everything in list that can do in set.
- faster to check in set than in list

List/Tuples/Strings are ordered
Dictionaries are mapped

my_dictionary = {key1 : data1, key2 : data2, }

name_to_age = {'Alice' : 12, "Bob" : 20, "Carol" : 35}
print (name_to_age.get('Alice')) / print (name_to_age['Alice'])

dictionary & set mutable

help(dict)

print (name_to_age)

random order (names + age) appear

name_to_age [id0]

id0	"Alice"
id1	12
id2	"Bob"
id3	20
id4	"Carol"
id5	35

id0 . id1
id2 . id3
id4 . id5

print (name_to_age["Bob"])

↓ 20

name_to_age["Alice"] = 13

name_to_age["Dave"] = 5

del name_to_age['Carol']

name_to_age[0] = "Hello"

```
for next_item in name_to_age:
    print (next_item + "name")   ← prints key
    print (next_name + "is" + str(name_to_age[next_name]) + "years old")
```

Lecture 8

my_object.method(data): The object has its own method and data

Class: type of an object

Object: instance of a class

Method: function which belongs to a class

instance
x = 7 — object

class ClassName():

def/var name pothole case
class name Camel Case

Class level docstring: gives basic description of what class does

python runs class by default without being called

my_first_object = MyCoolClass() — type @main

```
>>> my_first_object.var1 = 3
>>> print(my_first_object.var1)
3
```

my_first_object | id0

id0 → MyCoolClass { var1 | id1 }

id1 → 3

```
class MyCoolClass():

    def my_cool_method():
        print("Im in a method")

MyCoolClass.my_cool_method() -prints Im in a method

my_first_object = MyCoolClass().
my_first_object.my_cool_method(10)
```

first it sends object location, then the thing actually being passed. 10. 2 variables required in func

```
object_name.method_name(params)
```

⇓ - same

```
ClassName.method_name(object_name, params)

∴ my_first_object = MyCoolClass()

same [ my_first_object.my_cool_method(10)
       MyCoolClass.my_cool_method(my_first_object, 10)

def my_cool_method(self):
```

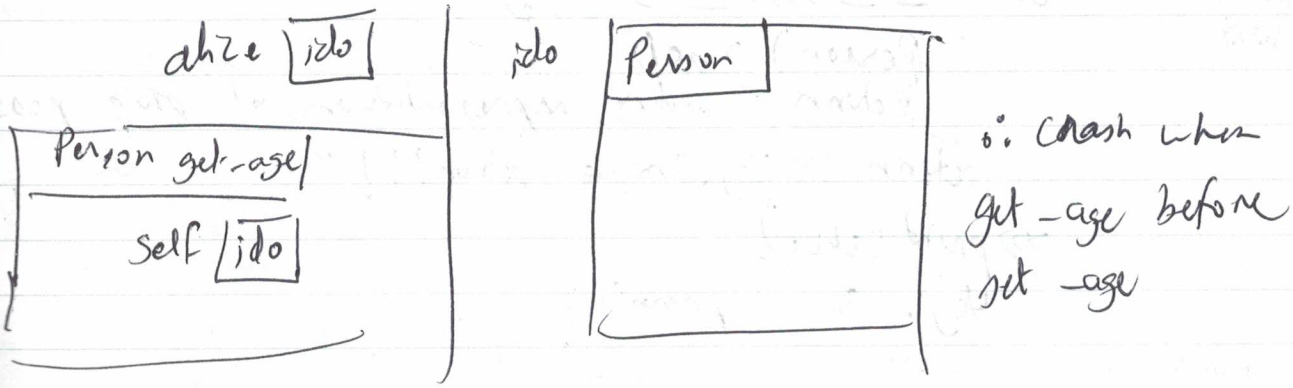
convention

```
class Person():
    "A class to represent a human being"

    def set_age(self, age):
        """(Person, int) -> None

        Set the age of this person to age
        REQ: age >= 0
        No examples required
        """
        self._age = self
```

```
alice = Person()
alice.set_age(35)
alice.age
~~self.age = self~~
35
```

All private into set self._age = age

∴ alice.set_age(30)
print(alice.age)
CRASHES



∴ crash when get_age before set_age

- `__init__`
  - Initialize: known as construction method
  - EVERY FUNCTION MUST HAV ONE

Ex: class person():

```
    def '''
        '''

        def __init__(self):?, name, age, gender
            '''(Person) -> NoneType
            Creats a new person
            '''
            print("I'm creating a new person")
```

ANYTHING WHERE YOU NEED TO SET UP UR VALUES CAN BE DONE HERE

```
            self.a_name = name
            self._age = age
```

- `__str__`
  - Returns what you want outputted when this class is changed to string

cant do → string

```
def __str__(self):
    '''(Person) -> self
    Return a string representation of this person
    '''
    return "Hey. Im a person!!!"
>>> print(alice)
Hey. Im a person!!!
```

Lecture 9

- **Association**
  - One object holds another object as variable

- "has a" relationship
- washing machine has access to shirt

Ex - A dog has a toy

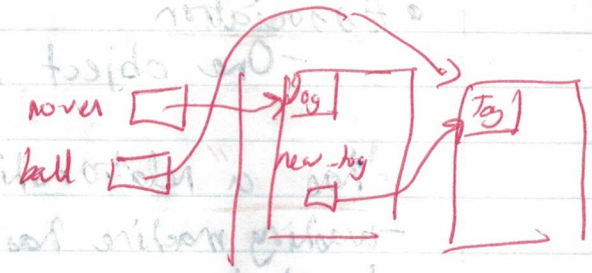
```
class Toy():
    """Dog toy makes sound when squeezed"""

    def __init__(self, sound):
        """Data type
        Description
        """
        self._sound = sound

    def squeeze(self):
        """Data type
        Description
        """
        print(self._sound)
```

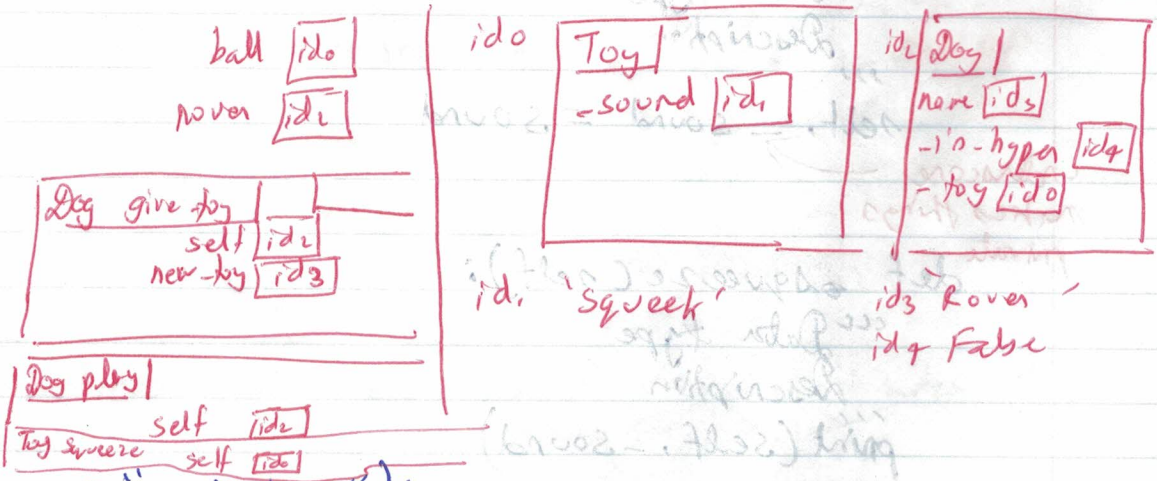
underscore makes things private

```
class Dog():
    "Description"
    def __init__(self, name, breed):
        """Data type
        Description
        """
        self._name = name
        self._io_hyper = breed in {"Breed name 1", "Breed name 2"}
```

```
ball = Toy('jingle')
ball.squeeze()
```

```
def give_toy(self, new_toy):
    """ Data type
        Description
    """
    self._toy = new_toy

def play(self)
    '''
    '''
    if (self._is_hyper):
        for i in range(100):
            self._toy.squeeeeze()
    else:
        self._toy.squeeze()
```



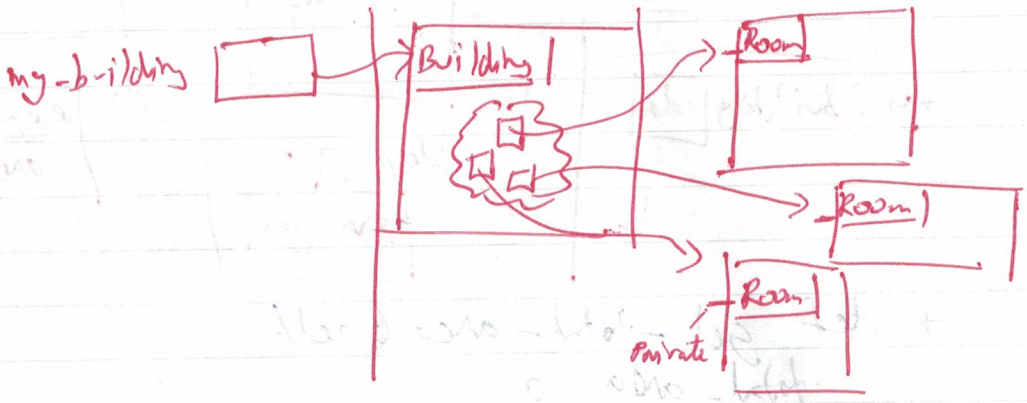



```
if (a__name__ = "__main__"):
    ball = Toy("squeek")
    rover = Dog("Rover", "Newfoundland")
    rover.give_toy(ball)
    rover.play()
```

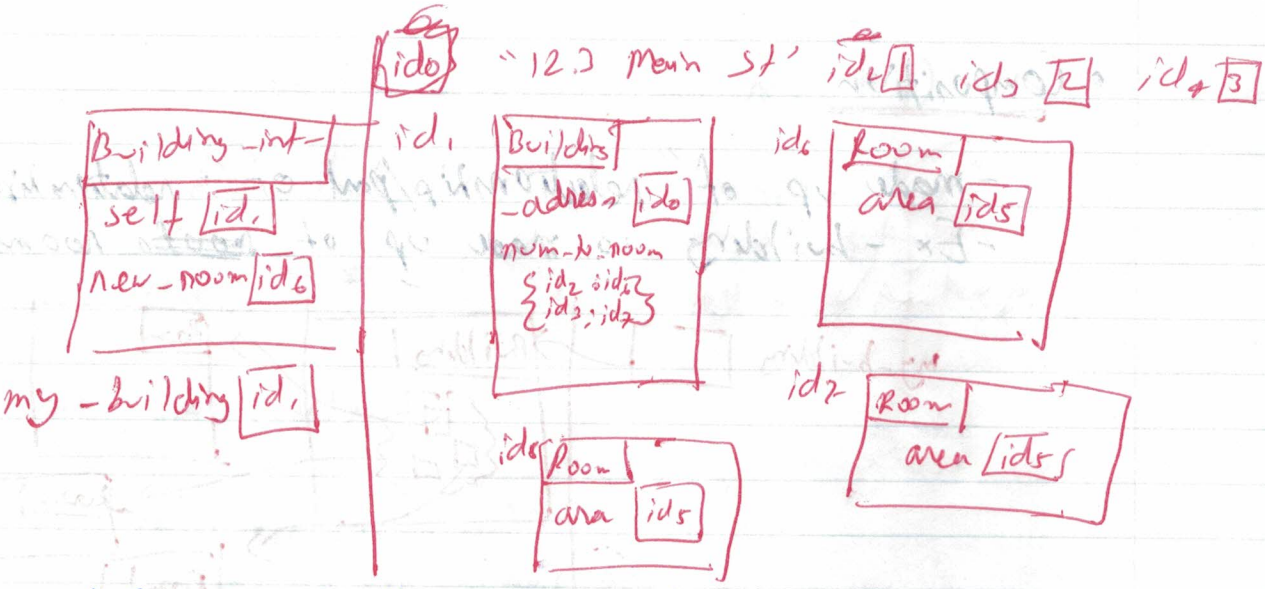
Dog can access any of the methods of the ball

- Composition

"made up of" relationship/part of relationship
- Ex - building is made up of ~~rooms~~ rooms




```
class _Room():
    """
    def __init__(self, length, width):
        """
        self._area = length * width

    def get_area(self):
        """(_Room) -> int
        """
        return self._area

class Building():
    """
    def __init__(self, address, num_rooms, room_length, room_width):
        """
        self._address = address
        self._num_to_room = {}
        for i in range(num_rooms):
            new_room = _Room(room_length, room_width)
            self._num_to_rooms[i] = new_room
* next_page
if (name_ == "__main__"):
    my_building = Building("123 Main st", 3, 10, 5)
```



* def get_total_area(self):
    total_area = 0
    for next_room_num in self.num_to_rooms:
        next_room = self.num_to_room(next_room_num)
        total_area += next_room.get_area()
    return (total_area)

- Inheritence
  - "is a" relationship
  - Ex: student is a person

class Person():
    def __init__(self, name, age):
    def get_name(self):
    def get_name(self):

```
class ParentClass():
    def method1(self):

class ChildClass(ParentClass):
    def method2(self):
if (name == .    ):
    child = childClass()
    child.method1()
     child.method()
```

* can chain

class GrandChildClass(ChildClass):

Lecture 10

black box - function
- can only see what goes in/comes out
- test major test areas + boundary cases
- testing can be done by external user

white box - see inside of box
- can test for weaknesses specific to the way implemented
- more focused testing

```
import unittest
import week10_functions as func        - file to be tested

class TestCommonChar(unittest.TestCase):
    def test_identical_single_char(self):
        self.assertEqual(
        func.common_chars('a', 'a'),
        (1, 1), "identical single char")
unittest.main(exit=False)
```

test have to be there. The rest is name

- One class per func to test
- Func name must start with "test"
- Expected return value
- message for when error occurs

Coverage testing - cover all possible scenarios
- when exhaustive testing too tedious, break up "test space" into areas
- pick representative example from each area
- pick examples from boundaries between areas

Adversal testing - try to break yours (or someone else's code)
- usually whitebox

Regression testing - when you make a change, check that you haven't introduced a bug to other code
- built new test cases on top of old, run everything

Testing levels

Unit - individual components
integration - putting components together
System - system as a whole
Acceptance - testing with users
Release - testing in the real world
- alpha (select group, expecting buggy code)
- beta (larger group, expect mostly working code)
- full release (it better be working by this point)

try:
    block of code
except:
    code to execute in case of ~~an~~ an error

---

try:

except ExceptionType:

ex. Zero division error, ValueError
found in error statement in shell

Similar to if = try
elif = except ExceptionType
else = except
else = else when no errors occur
finally = code that will be executed whether an exception raised or not

raises = raises a new exception
ex = raise ValueError("This is my message")

Create Exception

class BrianError(Exception)
    """
    """
raise BrianError("This is my message")

try block should have abt 2-4 lines