The Very First Obstacle of Programming

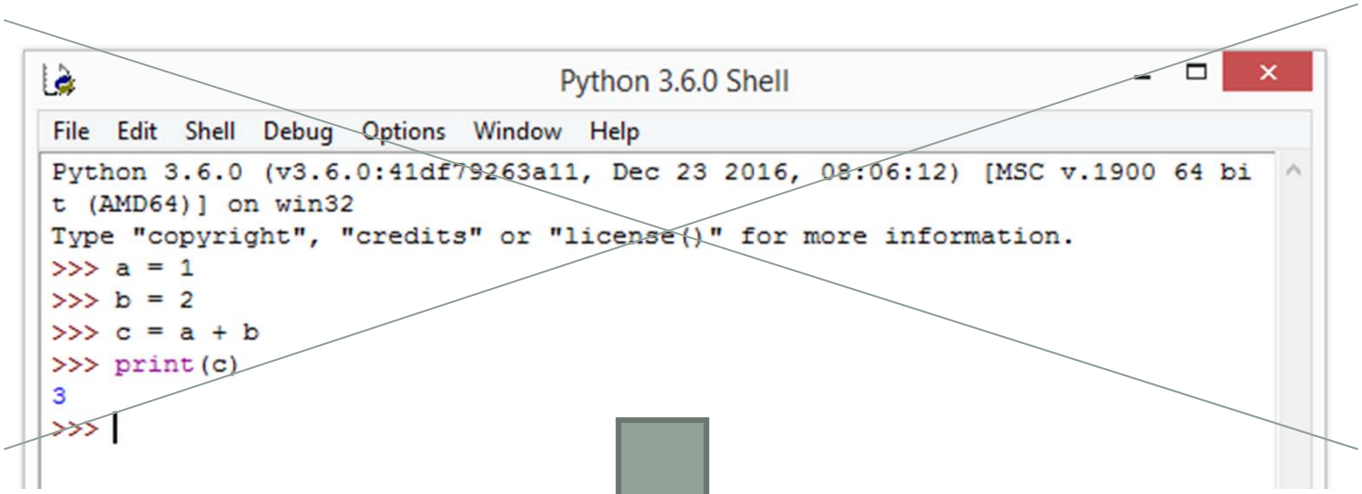
- Syntax Error
  - A syntax error is an error in the source code of a program. Since computer programs **must follow strict syntax** to compile correctly, any aspects of the code that do not conform to the syntax of the programming language will produce a syntax error.

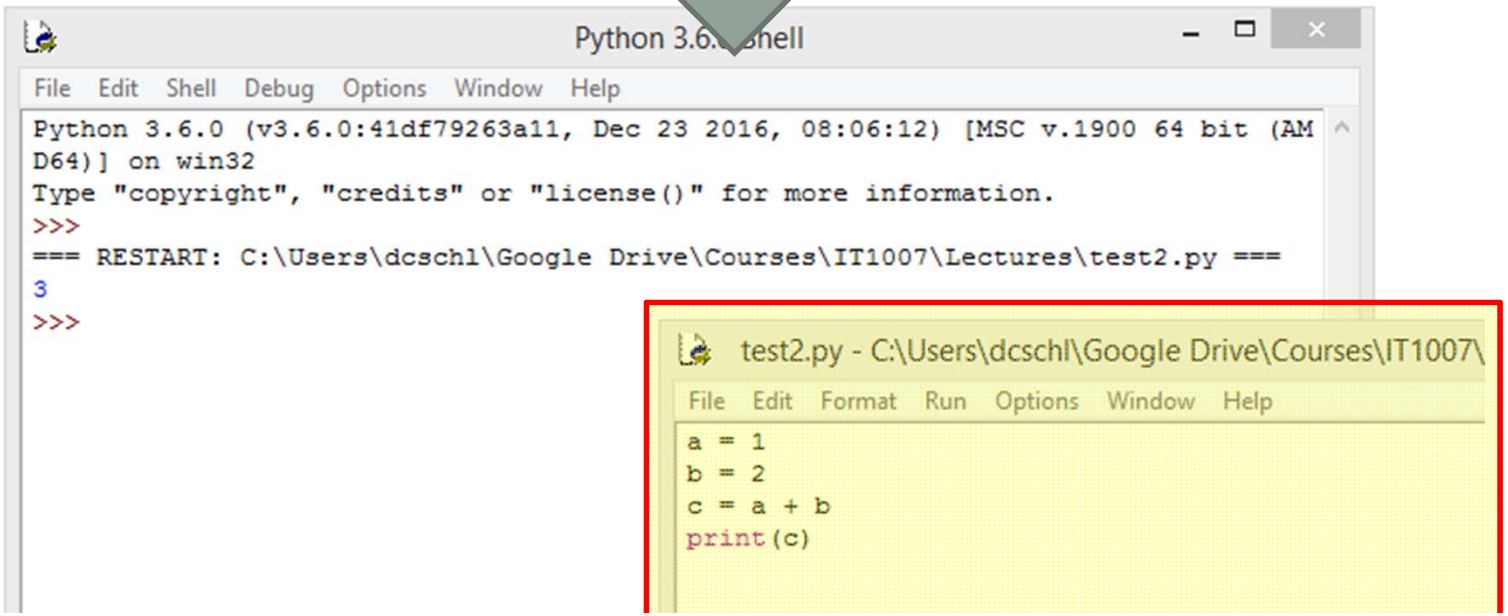
```
>>> x = 10
SyntaxError: invalid syntax
>>>
```

Sometime Errors are Fatal

- https://www.youtube.com/watch?v=VjJgiDuHlRw

Let's Move Out of the Console





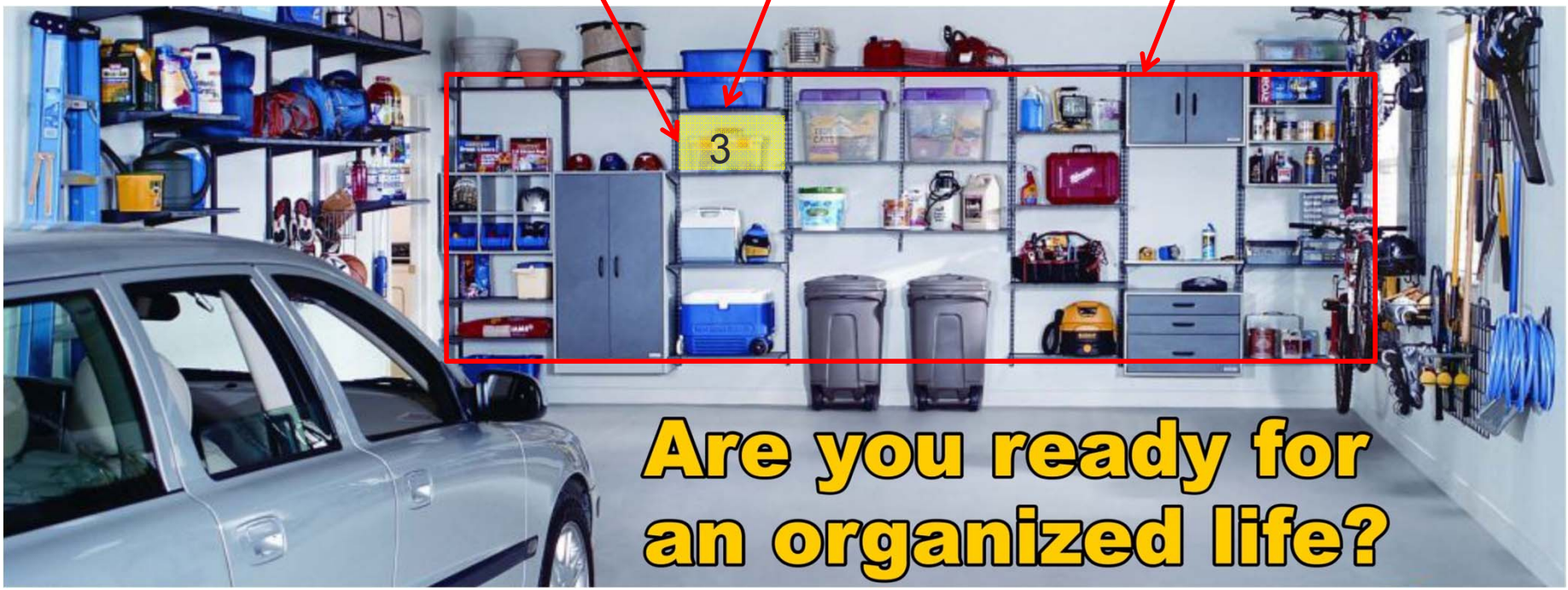
Variables

- For storage of data

```
>>> x = 3
```

This space is allocated and labeled as “x”. And we store ‘3’ inside

Computer Memory



Variable Naming

- Start with 'a'-'z' or 'A'-'Z' or '_'
- Contain only alphanumeric characters or '_'
- Case sensitive

```
X_1 != x_1
```

- Avoid reserved keywords e.g. `if`
- Python convention: lower case letters separated by '_'
  - e.g. `count_change`

Data Types

8, 45, 123	int
2.71828, 3.14159 , 1.0	float
True, False	bool
“it1007” ‘it1007’	str
	None

Variable Type

```
>>> x = 3
>>> name = “Alan”
```

This space is allocated and labeled as “x”. And we store ‘3’ inside. And can store integers only

This space is allocated and labeled as “name”. And we store “Alan” inside. And can store strings only

The function Type(…)

```
>>> type(123)
<class 'int'>

>>> type('123')
<class 'str'>

>>> type(None)
<class 'None'>
```

Type conversion

```
>>> str(123)
'123'

>>> float('45.2')
45.2

>>> int(23.8)
23

>>> int('cs1010s')
ValueError!
```

Assignments

Doesn't matter if it's quote or double quote

```
>>> abc = 18
>>> my_string = 'This is my string'
>>> x, y = 1, 2
```

Assignments

```
>>> x = 10
>>> x = 2
>>> x = 4
>>> print(x)

???
```

“=“ is different from our usual “equal” in math

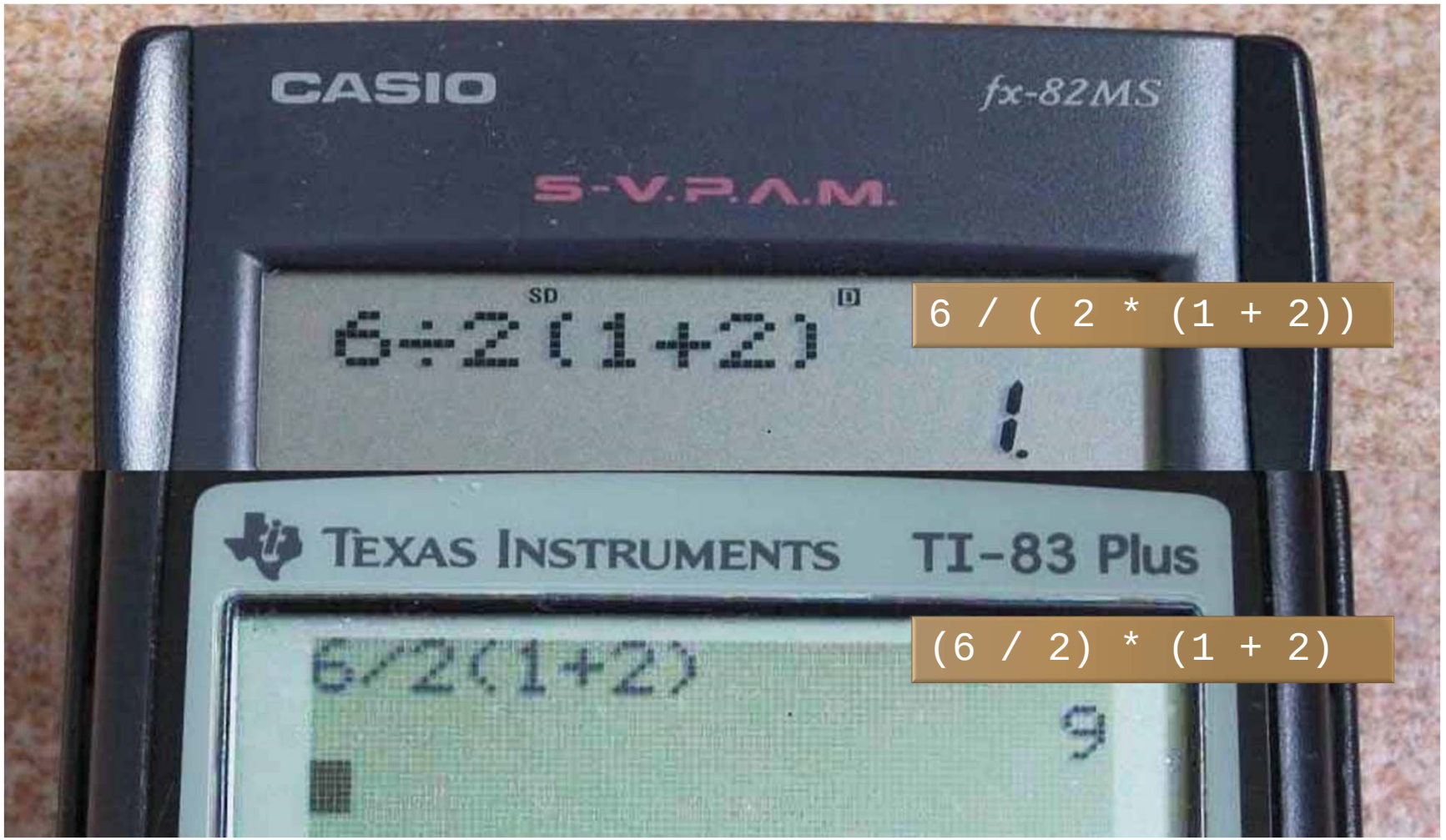
```
>>> a, b, c = 1, 2, 3
>>> a, b, c = c, b, a
>>> print(a, b, c)
???
```

Arithmetic: + - * / ** // %

```
>>> a = 2 * 3
>>> a
6

>>> 2 ** 3
8
```

```
>>> 11 / 3
3.6666666666666665

>>> 11 // 3
3

>>> 11 % 3
2
```

Operator Precedence



Python Operator Precedence

- 6/2*(1+2)
- 3*(1+2)
- 3*(1+2)
- 3*3
- 9

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'td>
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Boolean: Truth values

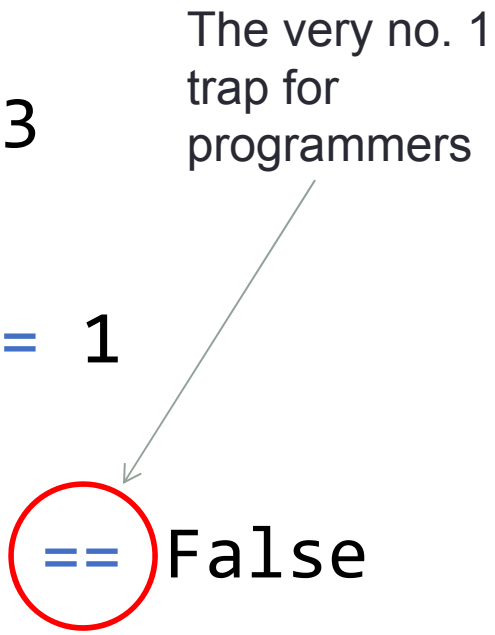
- Statements can be either True or False
- `2 > 1` is True
- `5 < 3` is False

Operators

- Comparison:

```
>>> 1 <= 10
True
>>> 5 > 15
False
>>> 5 <= 5
True
```

```
>>> 2 != 3
True
>>> '1' == 1
False
>>> False == False
True
>>> True != True
False
```



Operators

- Logic:

```
>>> True or False
True
>>> True and False
False
>>> not False
True
```

- a or b True if either a or b is True
- a and b True if both a and b are True
- not a True if a is not True

Truth Tables

A	NOT A
True	False
False	True

A OR B		A	
		True	False
B	True	True	True
	False	True	False

A AND B		A	
		True	False
B	True	True	False
	False	False	False

Truth Value Revisited

- Python has keywords `True` and `False`
- In Python 3.x, `True` and `False` will be equal to `1` and `0`
- Anything that is not `0` or `empty` will be evaluated as `True`
- Logic:

```
>>> True and 0
0
>>> not 'abc'
False
>>> 1 or 0
1
```

Strings

```
>>> s = 'ba'
>>> t = 'ck'
>>> s + t
'back'
>>> t = s + 'na' *
2
>>> t
'banana'
```

```
>>> 'z' in t
False
>>> 'bananb' > t
True
>>> 'banan' <= t
True
>>> 'c' < t
False
```

lexicographical ordering: first the first two letters are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two letters are compared, and so on, until either sequence is exhausted.

Strings

```
>>> w = ‘banana’
>>> s = w + w
>>> print(s)
‘bananabanana’
>>> s = w*3
>>> print(s)
‘bananabananabanana’
```

```
>>> s = (w+’ ‘)*2
>>> print(s)
‘banana banana ‘
```

Strings

- A String is a sequence of characters
- We can index a string, i.e.

```
>>> s = 'abcd'
>>> s[0]
'a'

>>> s[2]
'c'
```

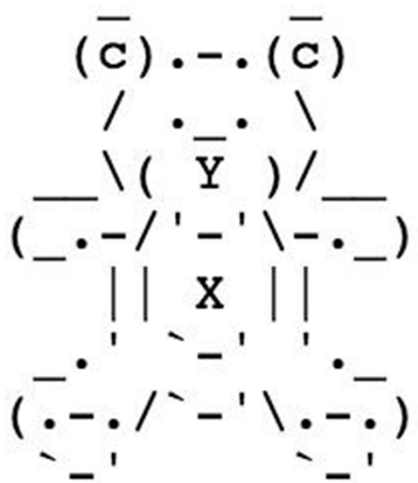
- The index of the first character is 0

Strings

```
c = "#"
s = " “

print(" ")
print("   *")
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(s*3 + c)
print(s*2 + c * 3)
print(s + c * 5)
print(c * 7)
print(s*3 + c)
```

```
   *
   #
  ###
 #####
   #
  ###
 #####
#######
   #
```

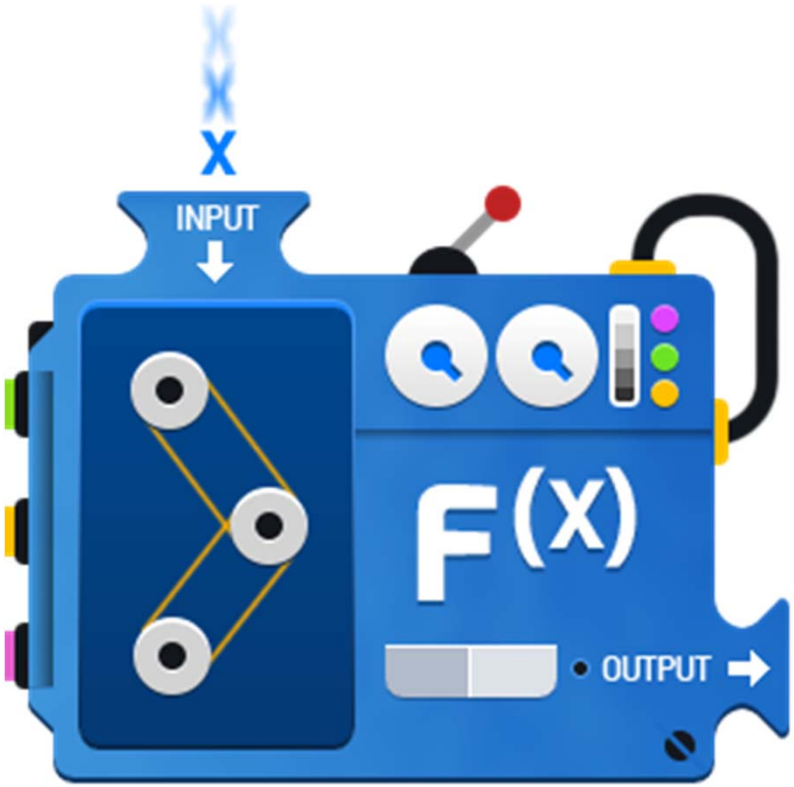
ASCII Art



FUNCTION ABSTRACTION

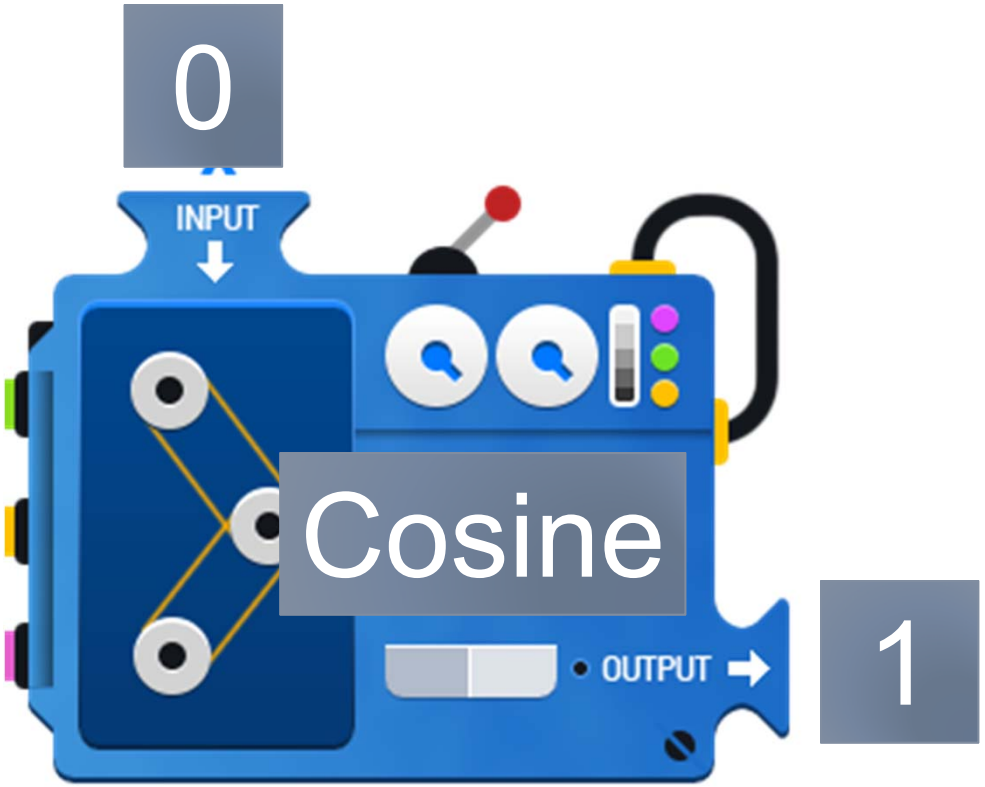
Recap: Functions

- A function is like a black box
  - You put in something for the input
  - And it will produce a new thing for the output

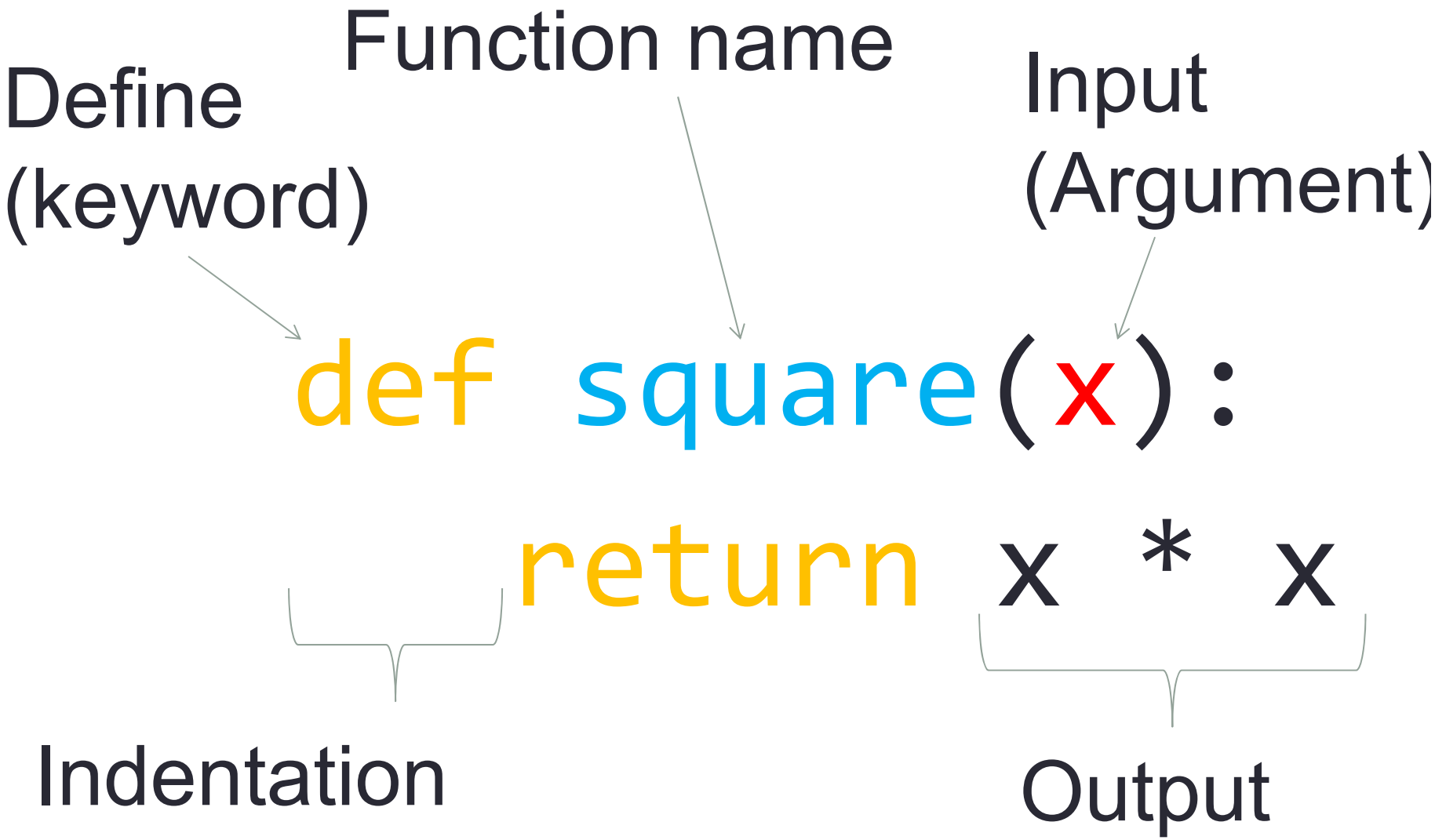


For example

- “Cosine” is a function
  - Input 0
  - Output 1



Let's Write Our Own Function!



Let's Write Our Own Function!

```
def square(x):
    return x * x
```

```
>>> square(3)
9
>>> square(square(2))
16
>>> square(3,4)
???
```

Another one….

```
def singHappyBirthdayTo(name):
    print('Happy birthday To You!')
    print('Happy birthday To You!')
    print('Happy birthday to ' + name + '~')
    print('Happy birthday to You!!!')
```

Indentation

```
>>> singHappyBirthdayTo('Alan')
Happy birthday To You!
Happy birthday To You!
Happy birthday to Alan~
Happy birthday to You!!!
```

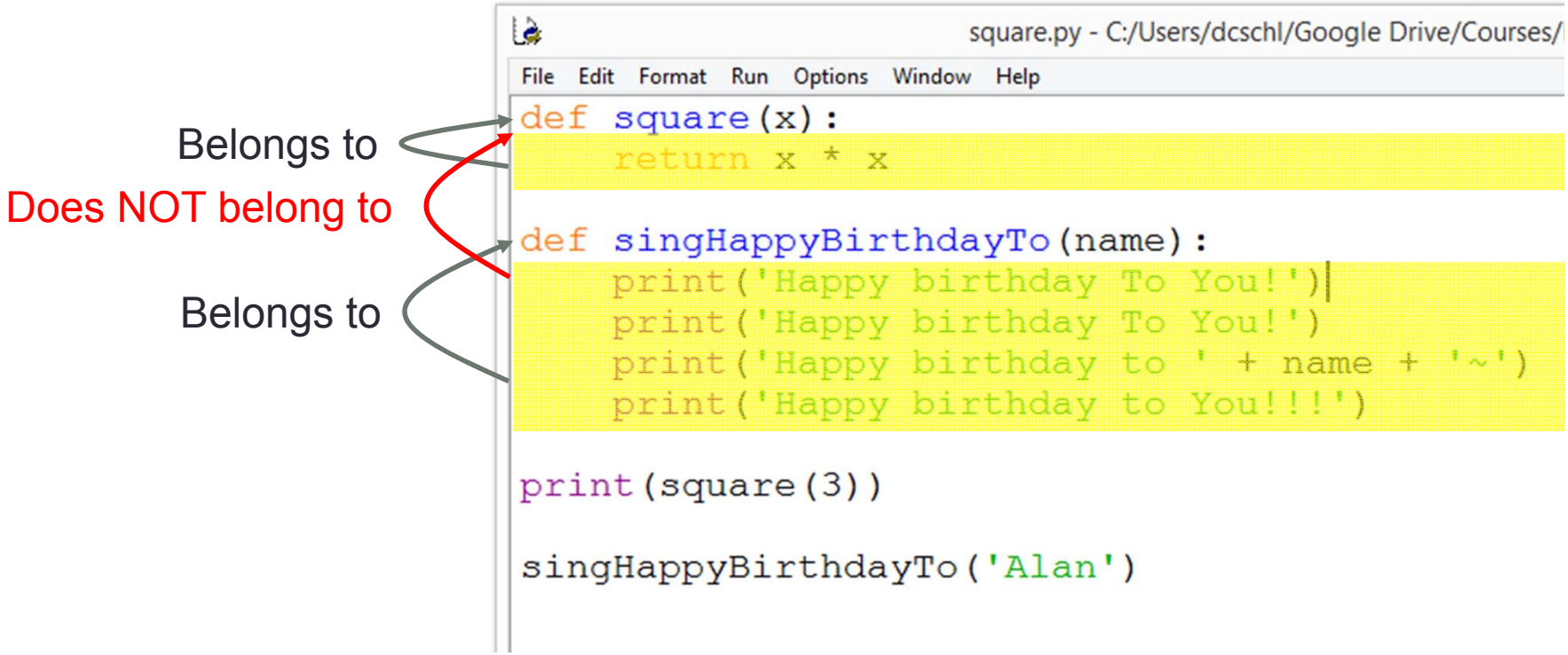
What if

A function

```
def singHappyBirthdayTo(name):
   print('Happy birthday To You!')
   print('Happy birthday To You!')
   print('Happy birthday to ' + name + '~')
print('Happy birthday to You!!!')
```

Execute an extra line **OUT** of the function

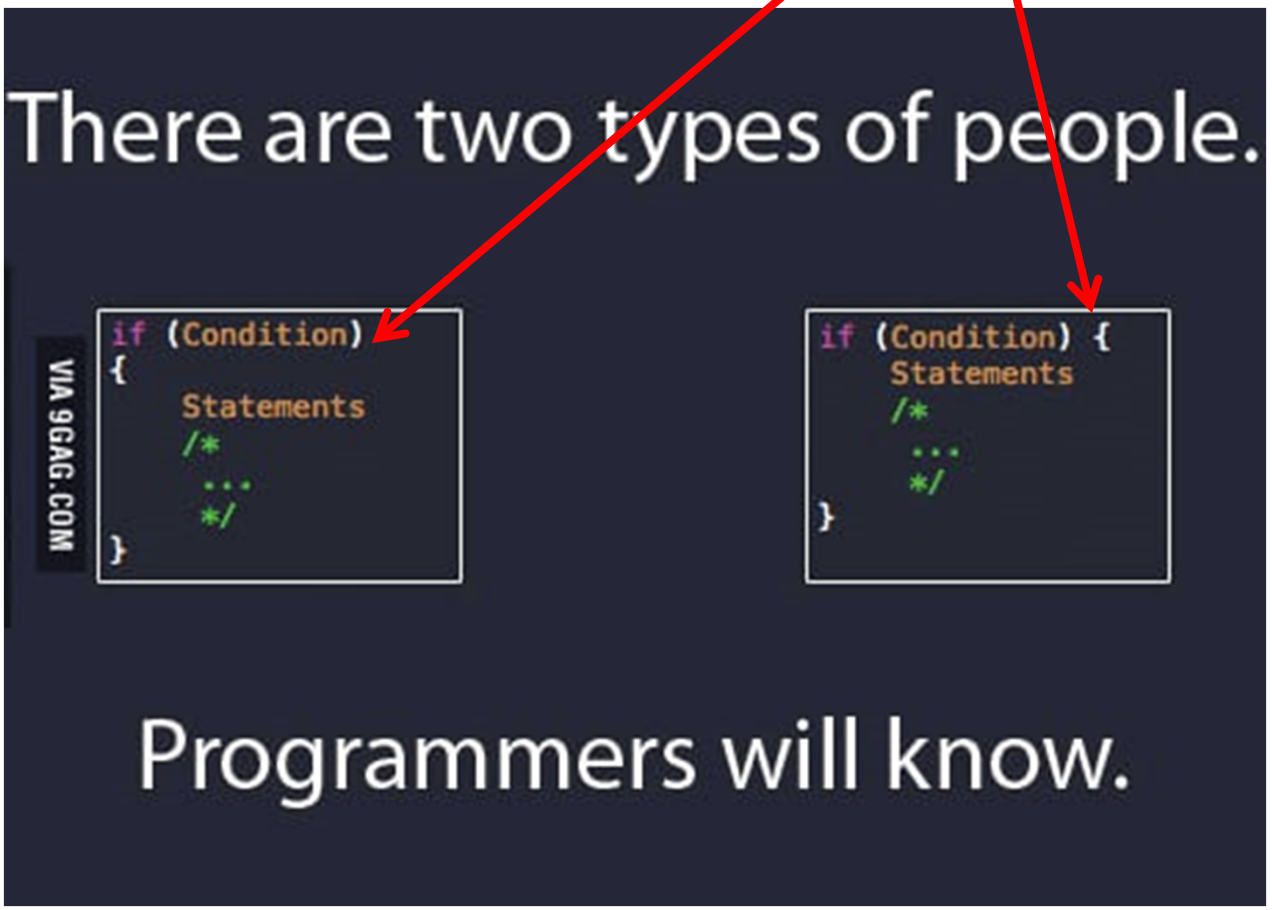
Indentation Matters



- This is a “Python thing”
  - Meaning, in other languages, usually spaces, tabs, new lines do not affect the code

C

- For C language, it doesn't matter



```php
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

Add Two Numbers

```
def addTwoNumbers() :
    a = int(input('Enter an integer:'))
    b = int(input('Enter another integer:'))
    print('The sum is:')
    print(a + b)
```

Keyboard input
(Built-in function)

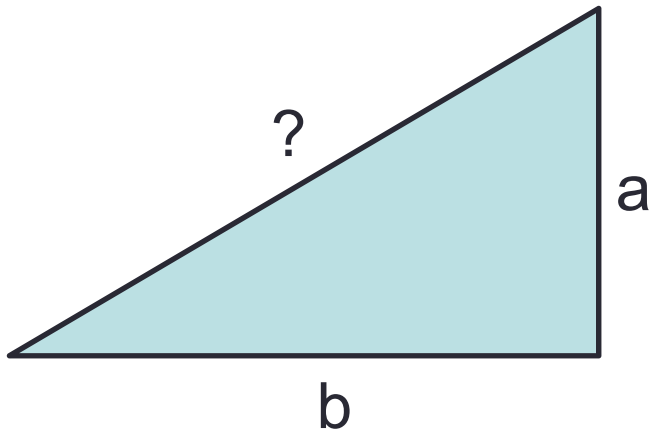
```
>>> addTwoNumbers()
Enter an integer:2
Enter another integer:3
The sum is:
5
>>>
```

Mix with Package Functions

A function from packages

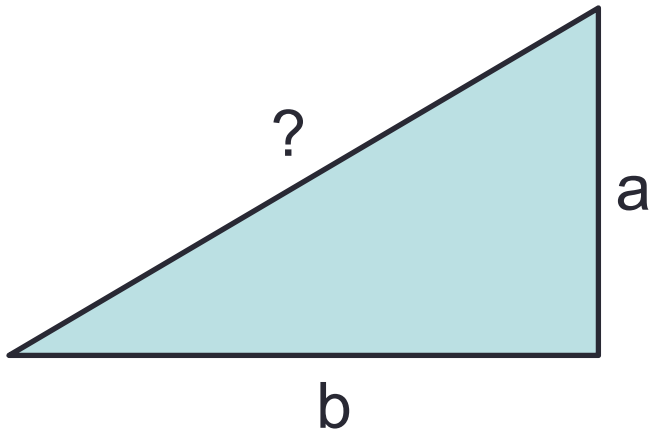
A function written by you

```
from math import sqrt

def hypotenuse(a, b):
    return sqrt(sum_of_squares(a, b))

hypotenuse(5, 12)
13
```



Compare:

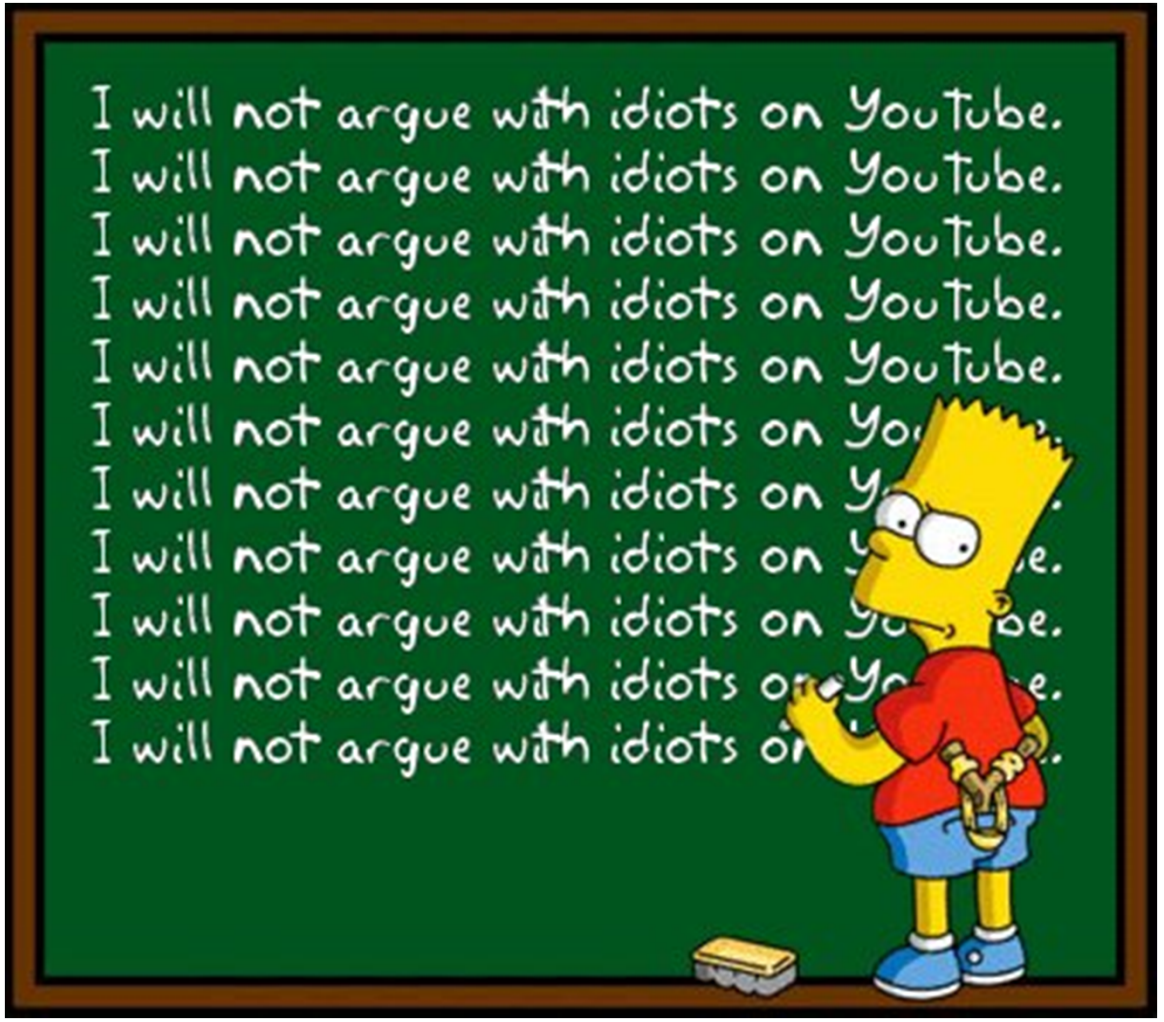
```
def hypotenuse(a, b):
    return sqrt(sum_of_squares(a, b))

def sum_of_squares(x, y):
    return square(x) + square(y)

def square(x):
    return x * x
```



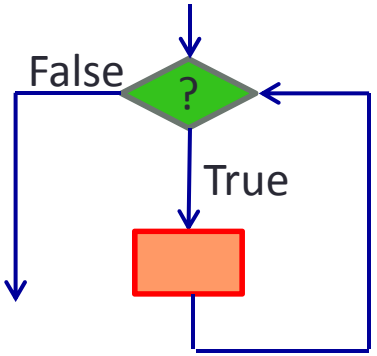
Versus:

```
def hypotenuse(a, b):
    return sqrt((a*a) + (b*b))
```

Repetition


I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.
I will not argue with idiots on YouTube.

Control Structure: Repetition



- While (a condition)
  - Do something

- For example

```
While (I am hungry)
  Eat a bun
```

- Again, can be more than one single instruction

```
While(I have money in bank)
  Take money out from bank
  Eat an expensive meal
  While(I have money in my wallet)
    Go Shopping
```

Repetition (Infinite)

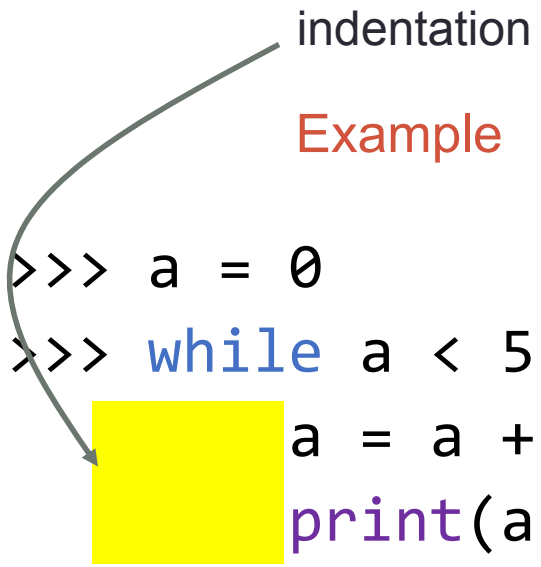
```
while True:
    print('Ah...')
```

```
a = 0
while > 0:
    a = a + 1
    print(a)
```

Repetition

Syntax

```
while <expr>:
    statement(s)
```



Example

```
>>> a = 0
>>> while a < 5:
        a = a + 1
        print(a)
1
2
3
4
5
```

Factorial

- The factorial n! is defined by

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

- Write a function for factorial?

```
def factorial(n):
    ans = 1
    i = 1
    while i <= n:
        ans = ans * i
        i = i + 1
    print(ans)
```

```
>>> factorial(3)
6
>>> factorial(6)
720
>>>
```

Repetition (nested)

inentation

Syntax

```
while <expr>:
    while <expr>:
        statement(s)
```

```
#
##
###
#
##
###
#
##
###
#
##
###
#
##
###
```

Example

```
def nestedWhile():
    i = 0
    while i < 5:
        i += 1
        j = 0
        while j < 3:
            j += 1
            print ('#' * j)
```

Repetition, a Very Common Pattern

9 out of 10 times you will do

```
>>> a = 0
>>> while a < N:
        a = a + 1
        do something
```

For loop

```
for i in range(0,N):
    do something
```

Another Repetition Flow Control: “For”

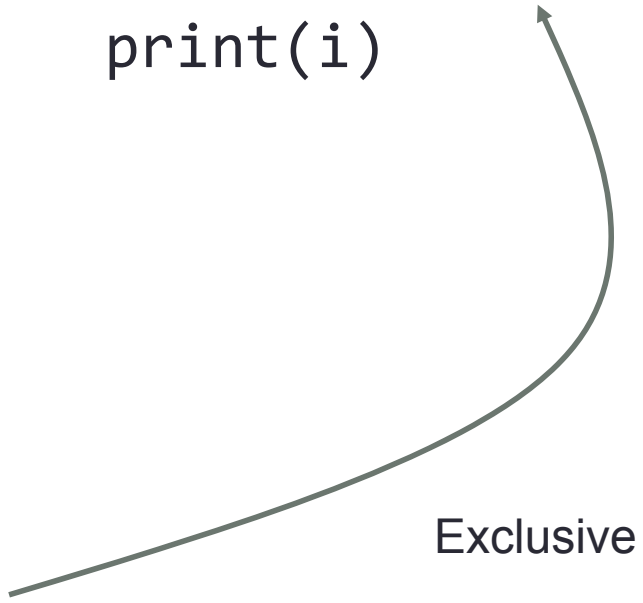
Syntax

```
for i in range(n,m):
    statement(s)
```

Example

```
for i in range(0,5):
    print(i)
```

```
0
1
2
3
4
```



Another Repetition Flow Control: “For”

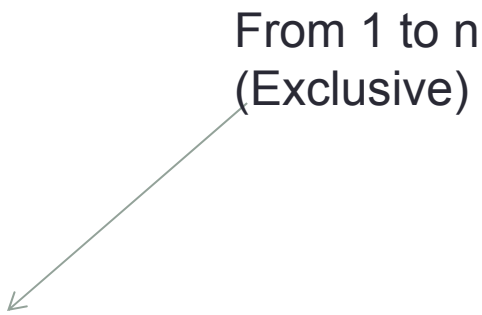
Example

```
for i in range(0,5):
      print(i)


0
1
2
3
4
```

Interpreted as

```
i=0
print(i)
i=1
print(i)
i=2
print(i)
i=3
print(i)
i=4
print(i)
```

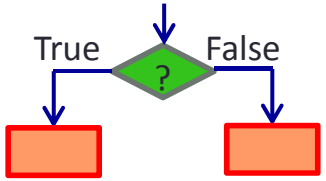
Factorial again




```
>>> def factorial(n):
      ans = 1
      for i in range(1,n+1):
            ans *= i
      return ans

>>> factorial(5)
120
>>>
```

Making Choices

Control Structure: Selection




```
If (a condition is true)
  Do A
Else
  Do B
```

Can be MORE THAN one single instruction

- For example:

```
If (I have $1000000000000000)
  Buy a car
  Eat a lot of buffets
  Go travel
  Quit NUS!
Else
  Be good and study
```

Conditional

Syntax

```
if <expr>:
    statement(s)
```

Example

```
>>> my_money = 1000
>>> if my_money > 0:
        print('Good')
'Good'
```

Conditional

Syntax

```
if <expr>:
    statement(s)
```

indentation

Example

```
>>> my_money = 1000
>>> if my_money > 0:
        print('Good')
        print('Good')
        print('Good')

'Good'
'Good'
'Good'
```

Conditional

Syntax

```
if <expr>:
    statement(s)
else:
    statement(s)
```

Example

```
>>> my_account = 1000
>>> if my_account > 0:
        print('rich')
else:
        print('broke')
'rich'
```

Conditional (Nested)

Syntax

```
if <expr>:
    if <expr>:
        statement(s)
```

Example

```
a = 4
if a < 10:
    if a < 1:
        print('Here')
```

Print nothing

Conditional

Syntax

```
if <expr>:
    statement(s)
else:
    statement(s)
```

Example

```
>>> my_account = 1000
>>> if my_account < 0:
        print('poor')
    else:
        if my_account > 1:
            print('v rich')

v rich
```

Clumsy

Conditional

Syntax

```
if <expr>:
    statement(s)
elif <expr>:
    statements(s)
else:
    statement(s)
```

Example

```
>>> a = -3
>>> if a > 0:
        print('yes')
elif a == 0:
        print('no')
else:
        print('huh')
'huh'
```

Conditional

Syntax

```
if <expr>:
    statement(s)
elif <expr>:
    statements(s)
elif <expr>:
    statements(s)
else:
    statement(s)
```

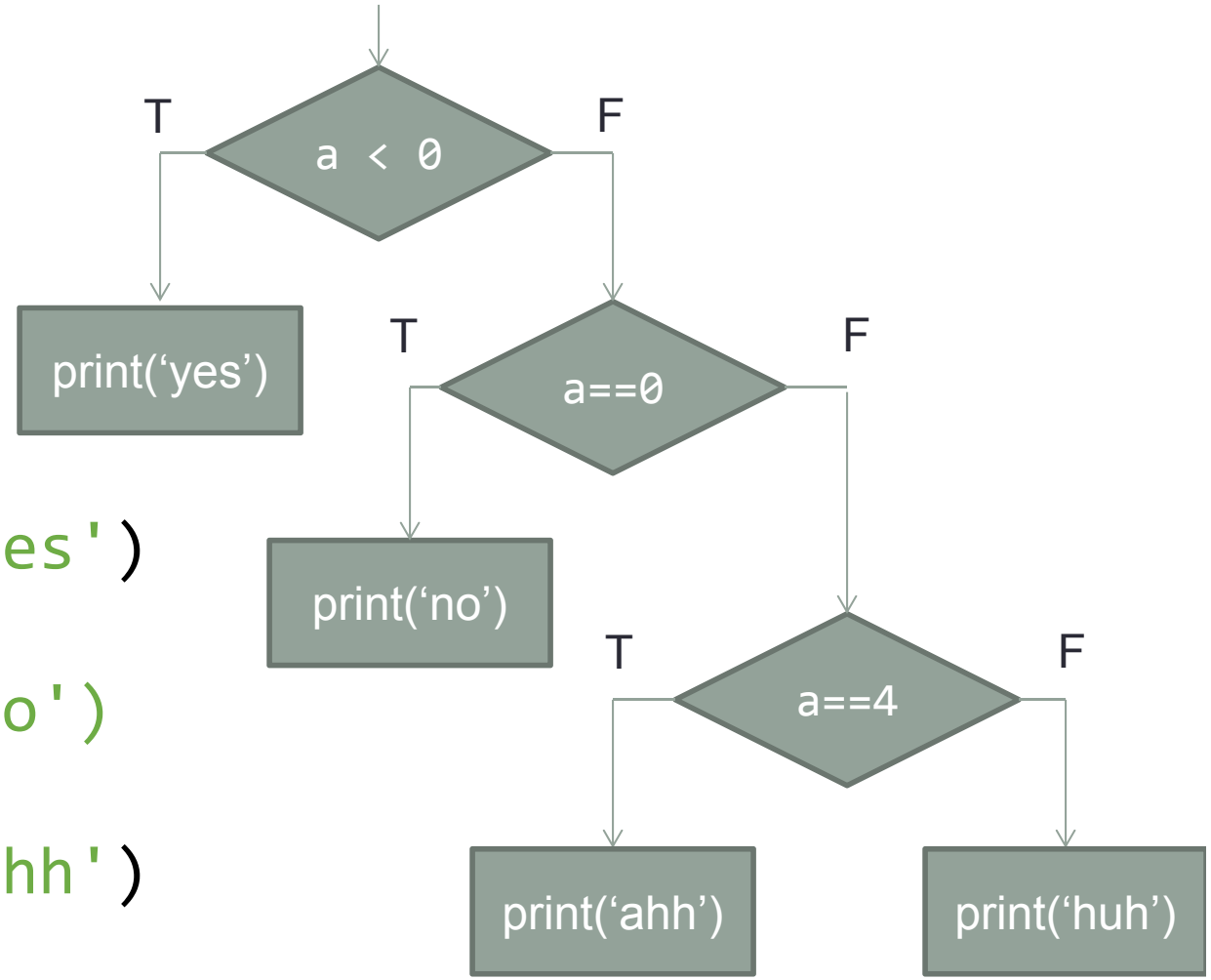
Can be many

Example

```
>>> a = 4
>>> if a > 0:
        print('yes')
elif a == 0:
        print('no')
elif a == 4:
        print('ahh')
else:
        print('huh')
'yes'
```

e.g.

```
>>> a = 4
>>> if a < 0:
        print('yes')
elif a == 0:
        print('no')
elif a == 4:
        print('ahh')
else:
        print('huh')
'ahh'
```

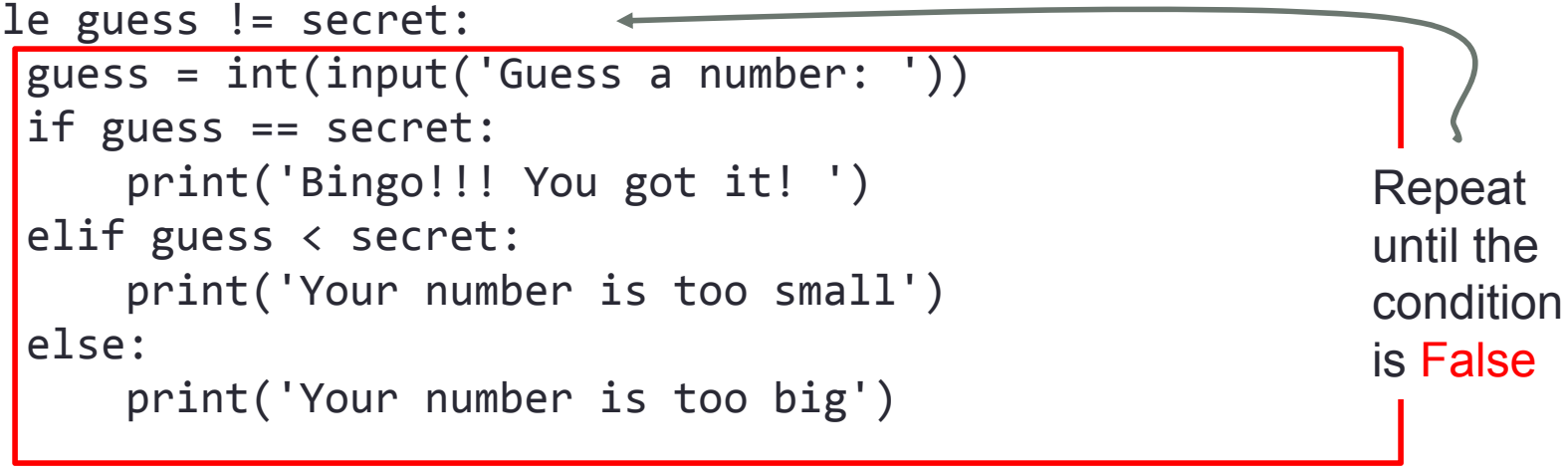


Let's play a game

```
>>> guessANum()
I have a number in mind between 0 and 99
Guess a number: 50
Too big
Guess a number: 25
Too big
Guess a number: 12
Too big
Guess a number: 6
Too small
Guess a number: 9
Too big
Guess a number: 7
Bingo!!!
>>>
```

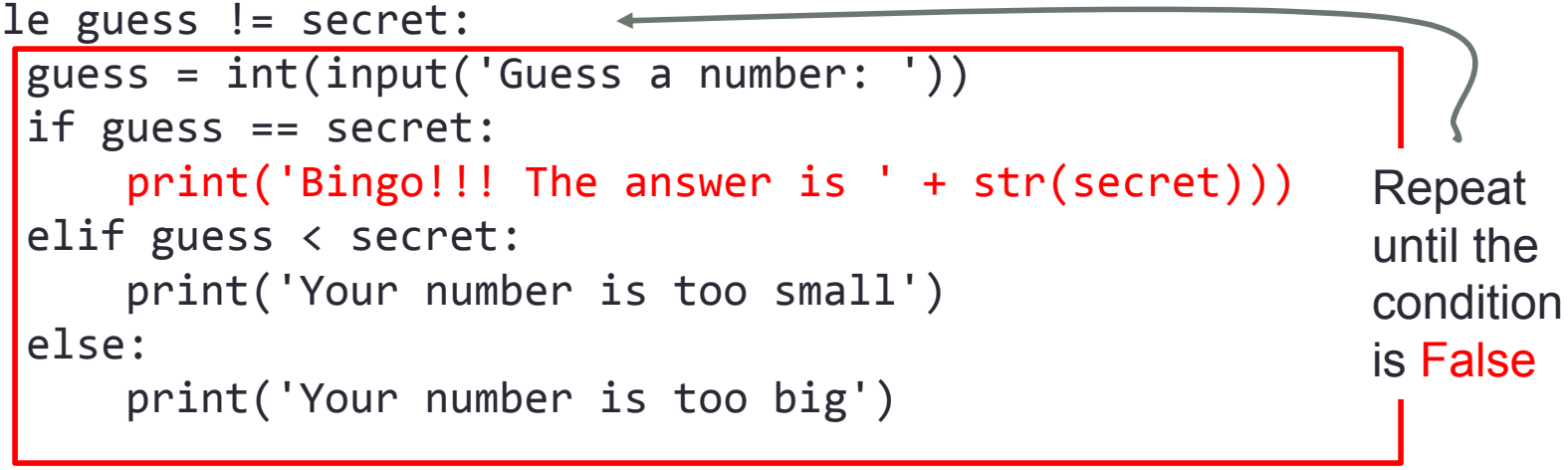
guessANum.py

```
import random

def guessANum():
    secret = random.randint(0,99)    # 0 <= secret <= 99
    guess = -1
    print('I have a number in mind between 0 and 99')
    while guess != secret:
        guess = int(input('Guess a number: '))
        if guess == secret:
            print('Bingo!!! You got it! ')
        elif guess < secret:
            print('Your number is too small')
        else:
            print('Your number is too big')

guessANum()
```



guessANum.py

```
import random

def guessANum():
    secret = random.randint(0,99)    # 0 <= secret <= 99
    guess = -1
    print('I have a number in mind between 0 and 99')
    while guess != secret:
        guess = int(input('Guess a number: '))
        if guess == secret:
            print('Bingo!!! The answer is ' + str(secret)))
        elif guess < secret:
            print('Your number is too small')
        else:
            print('Your number is too big')

guessANum()
```



Can you spot the difference?

Example 1

```
def foo():
  if True:
    if False:
        print(1)
    else:
        print(2)
```

Example 2

```
def foo():
  if True:
    if False:
      print(1)
  else:
    print(2)
```

Tips

- A “while” or “if” block starts with a colon “:”
- Remember
  - When there is a colon, there are indentations
  - When there are indentations, before these there is a colon
- The inclusive/exclusive range is a pain