# C10552: Intro to Computation

Lecture 1
Jul 10, 2016

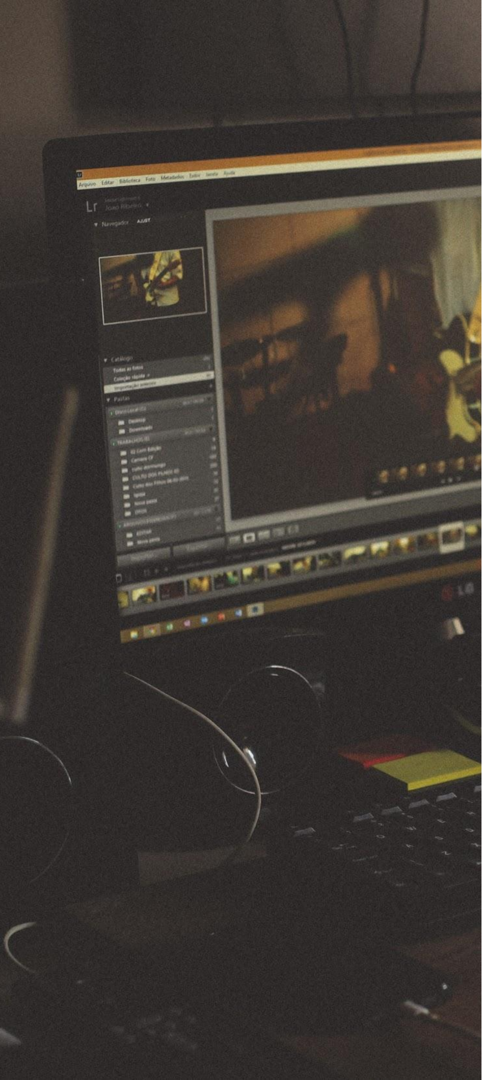
## Welcome!

C10552: Intro to Computation

Weekly lectures 1pm

HW: assigned weekly, optional, somewhat open-ended

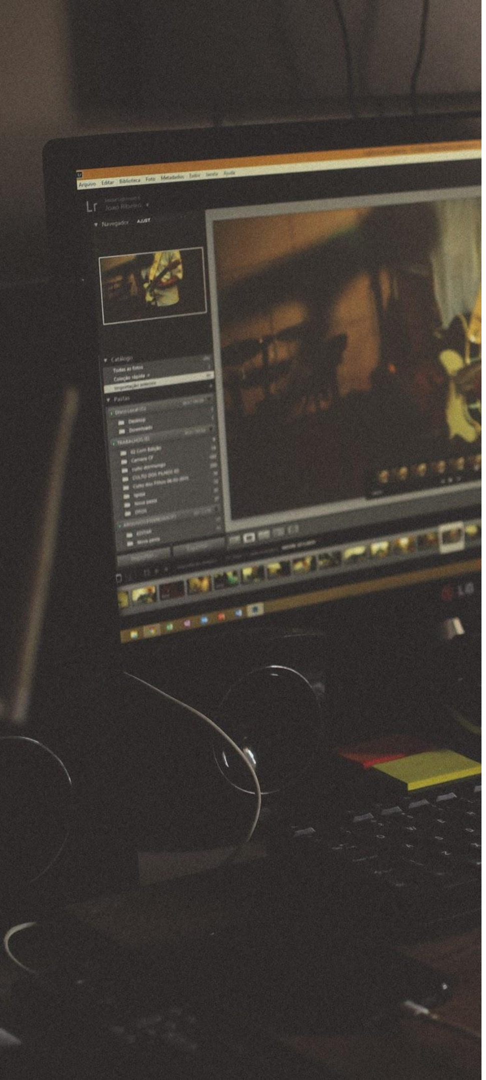
Submit to oceliker@mit.edu with subject [10552HW] for feedback



## Computers

**1.** A person who makes calculations or computations; a calculator, a reckoner; *spec*. a person employed to make calculations in an observatory, in surveying, etc. Now chiefly *hist*.

*from the Oxford English Dictionary*

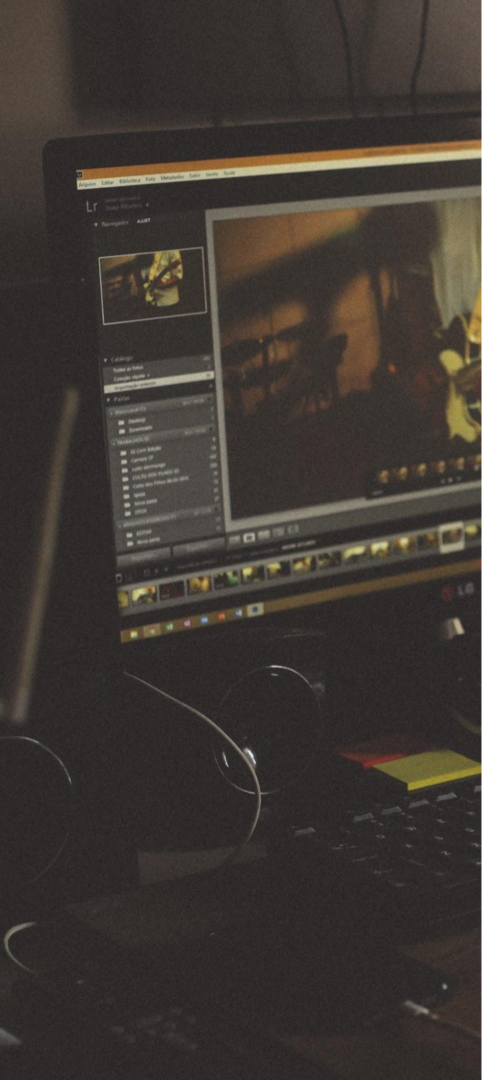


## Computers

**1.** A person who makes calculations or computations; a calculator, a reckoner; *spec*. a person employed to make calculations in an observatory, in surveying, etc. Now chiefly *hist*.

**3.** An electronic device (or system of devices) which is used to store, manipulate, and communicate information, perform complex calculations, ... and is capable of receiving information (data) and of processing it in accordance with variable procedural instructions (programs or software)...

*from the Oxford English Dictionary*

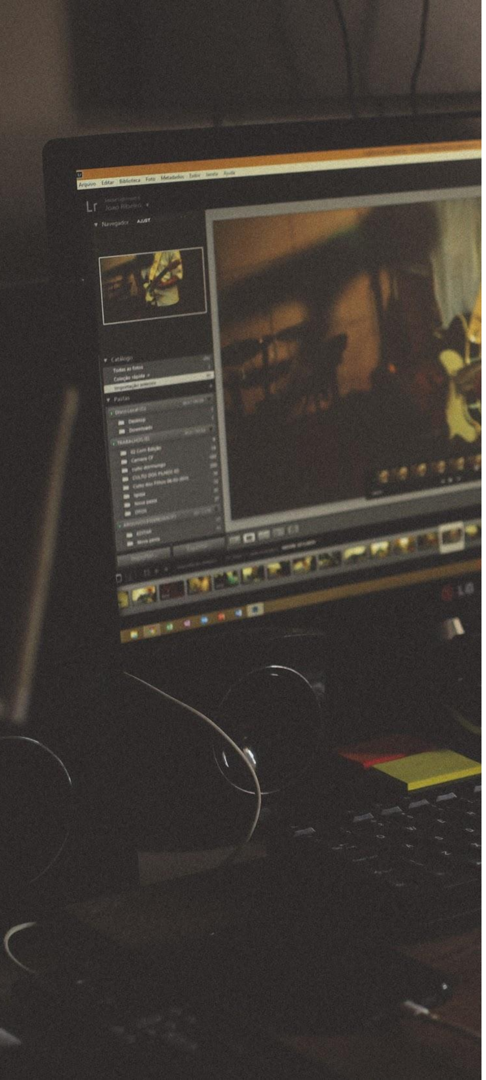


## Computers

**1.** A person who makes calculations or computations; a calculator, a reckoner; *spec*. a person employed to make calculations in an observatory, in surveying, etc. Now chiefly *hist*.

**3.** An electronic device (or system of devices) which is used to store, manipulate, and communicate information, perform complex calculations, ... and is capable of receiving information (data) and of processing it in accordance with variable procedural instructions (programs or software)...

*from the Oxford English Dictionary*

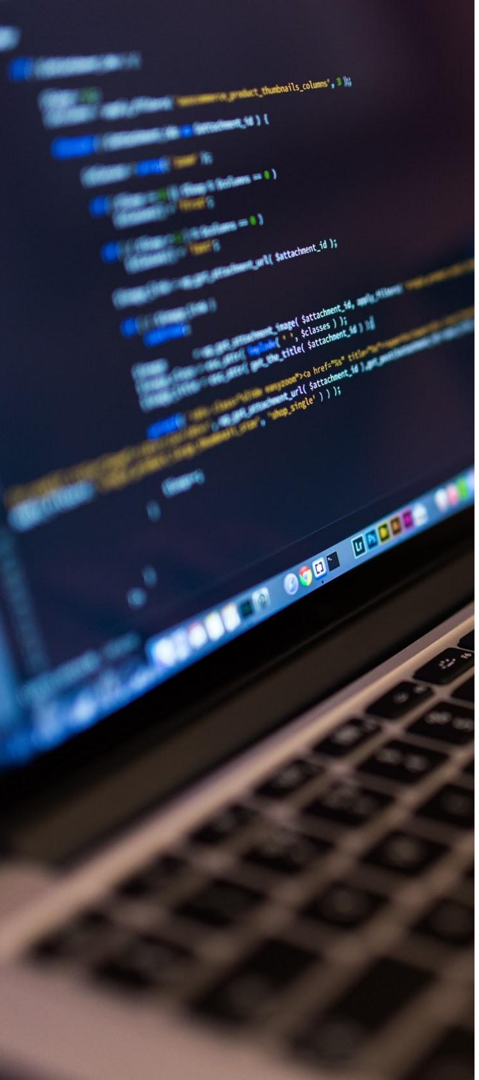


## Computers

**1.** A person who makes calculations or computations; a calculator, a reckoner; *spec*. a person employed to make calculations in an observatory, in surveying, etc. Now chiefly *hist*.

**3.** An electronic device (or system of devices) which is used to store, manipulate, and communicate information, perform complex calculations, ... and is capable of receiving information (data) and of processing it in accordance with variable procedural instructions (programs or software)...

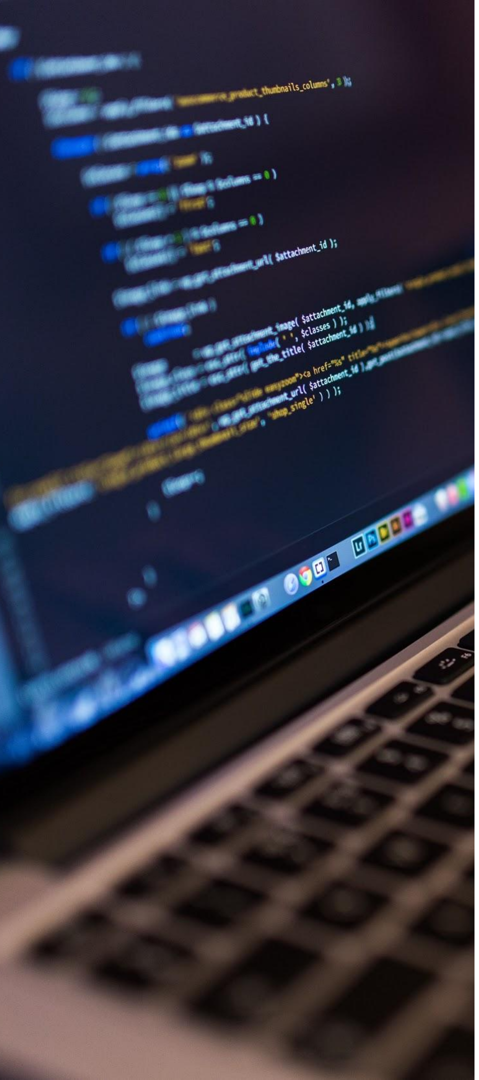
*from the Oxford English Dictionary*



## Variable procedural instructions

- aka “programs” or “scripts” or “code”
- A set of sequential commands to the computer

```
Let a = 5, b = 4, c = 2.
Compute a * b, call this d.
Compute d * c, call this e.
Display e on the screen.
```

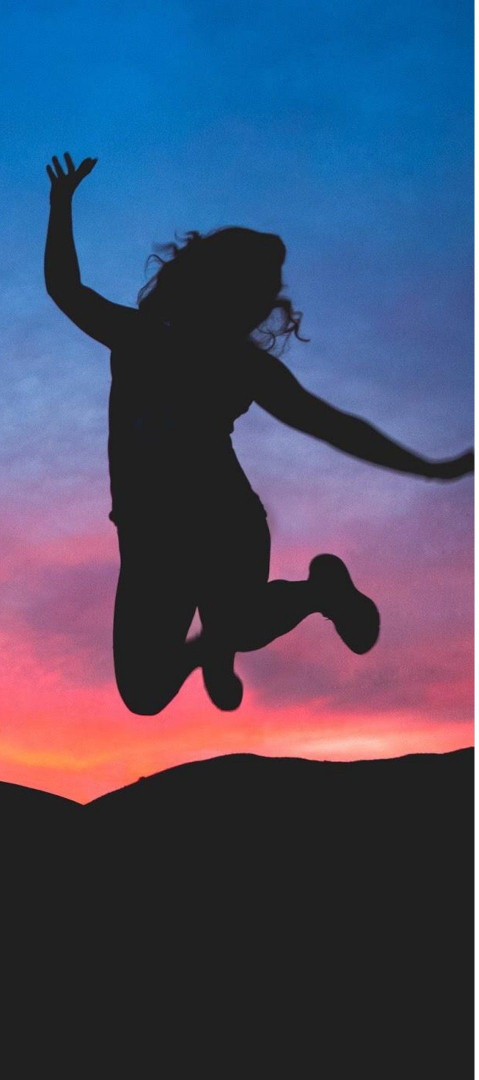


## Variable procedural instructions

- aka “programs” or “scripts” or “code”
- A set of sequential commands to the computer

```
Let a = 5, b = 4, c = 2.
Compute a * b, call this d.
Compute d * c, call this e.
Display e on the screen.
```

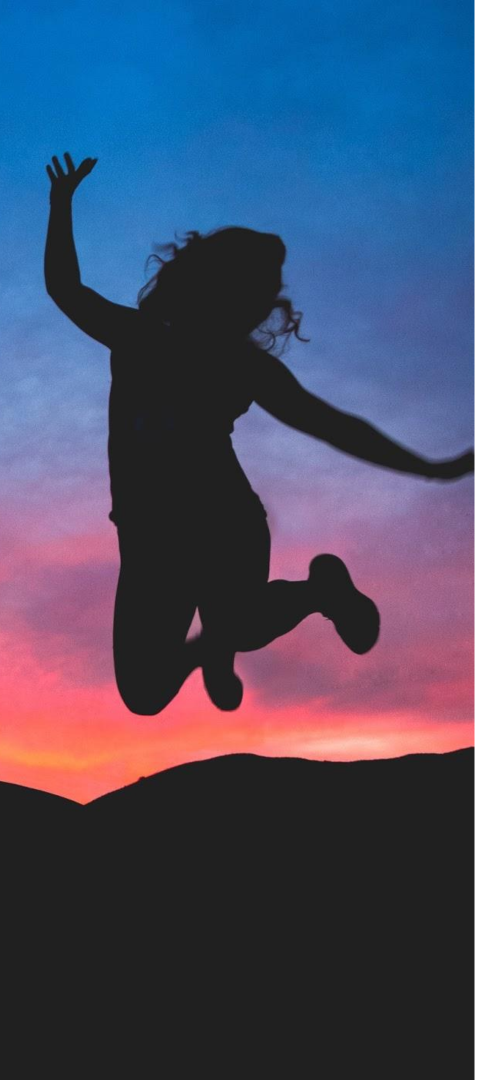
```
a = 5; b = 4; c = 2
d = a * b
e = d * c
print e
```



## Why do we like computers?

- … because they compute! :)
- (really, really fast)

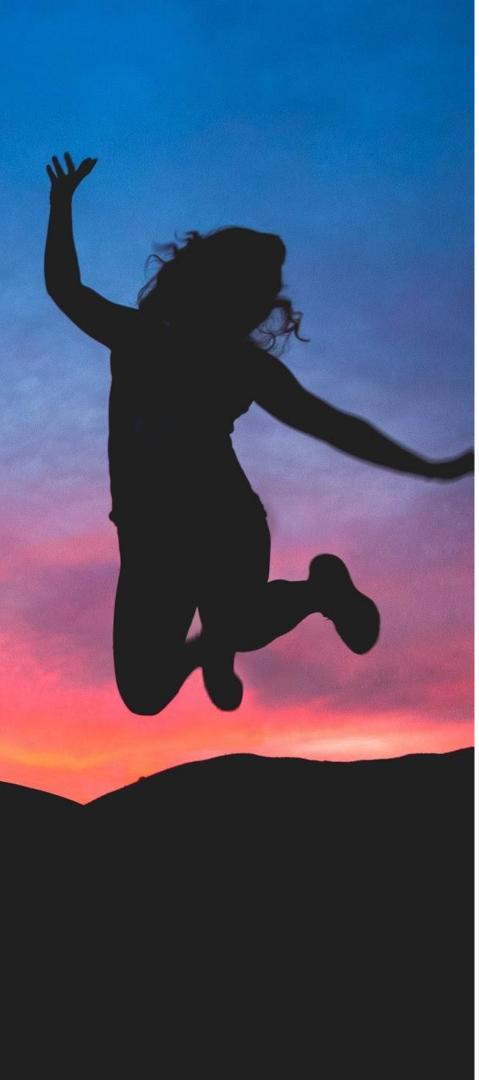
```
a = 5; b = 4; c = 2
d = a * b
e = d * c
print e
```

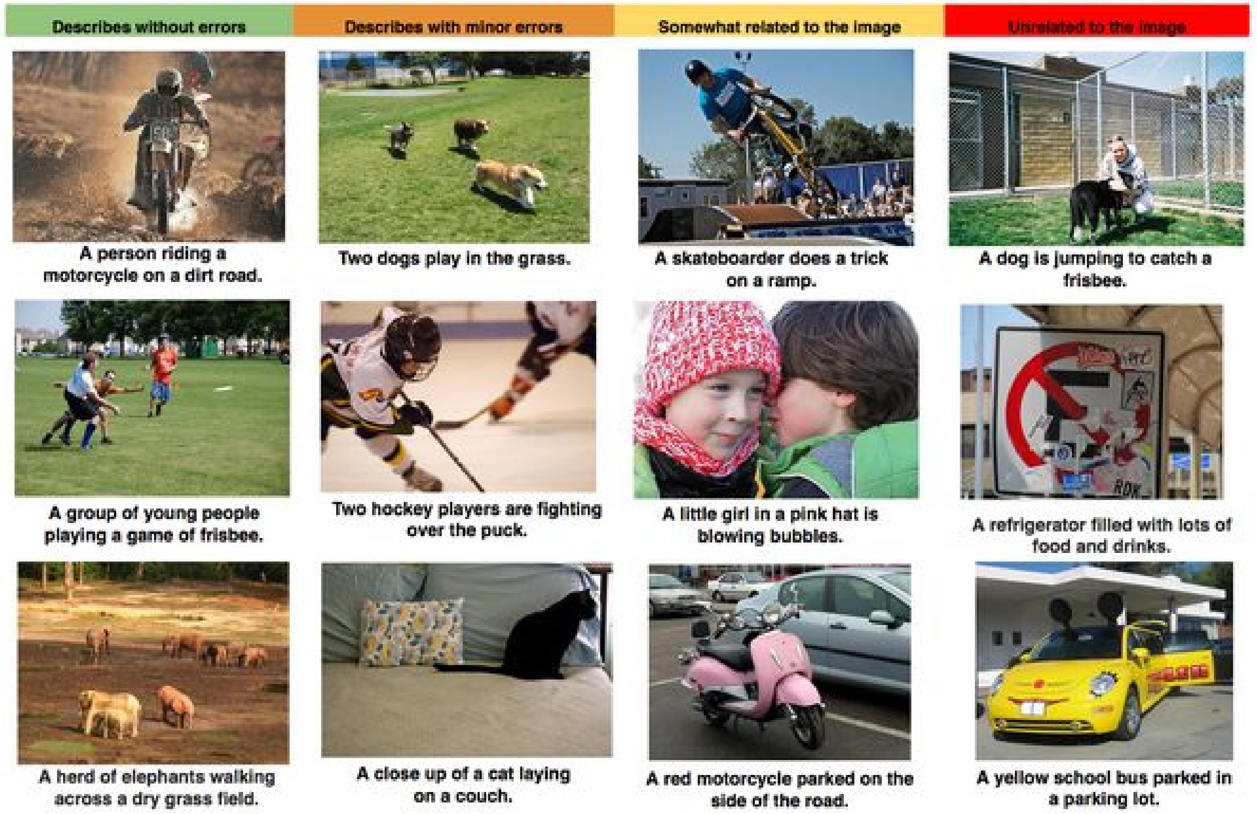


## Why do we like computers?

- ... because they compute! :)
- (really, really fast)

```
a = 5582; b = 41105; c = 24867221
d = a * b
e = d * c
print e
```

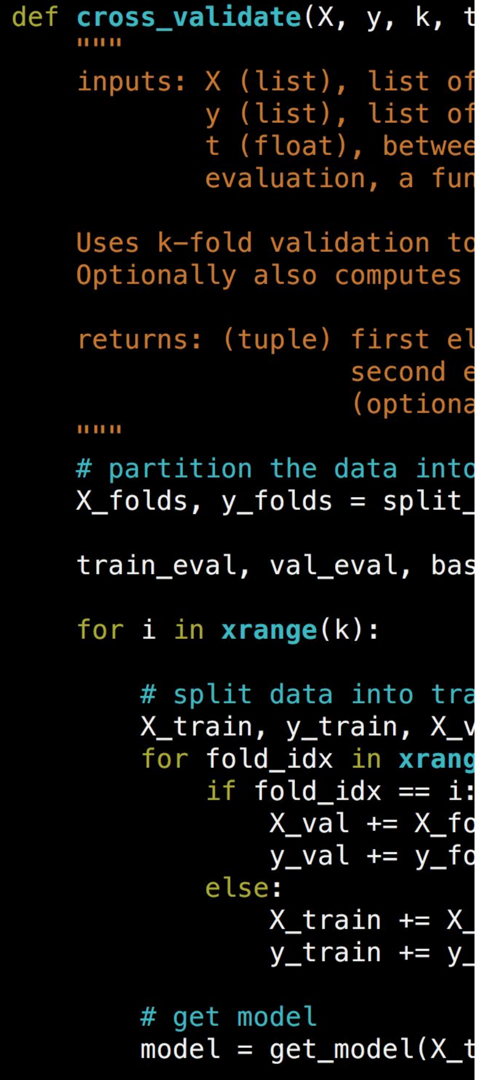




| Describes without errors | Describes with minor errors | Somewhat related to the image | Unrelated to the image |
|---|---|---|---|
| A person riding a motorcycle on a dirt road. | Two dogs play in the grass. | A skateboarder does a trick on a ramp. | A dog is jumping to catch a frisbee. |
| A group of young people playing a game of frisbee. | Two hockey players are fighting over the puck. | A little girl in a pink hat is blowing bubbles. | A refrigerator filled with lots of food and drinks. |
| A herd of elephants walking across a dry grass field. | A close up of a cat laying on a couch. | A red motorcycle parked on the side of the road. | A yellow school bus parked in a parking lot. |

*A selection of evaluation results, grouped by human rating.*

https://research.googleblog.com/2014/11/a-picture-is-worth-thousand-coherent.html

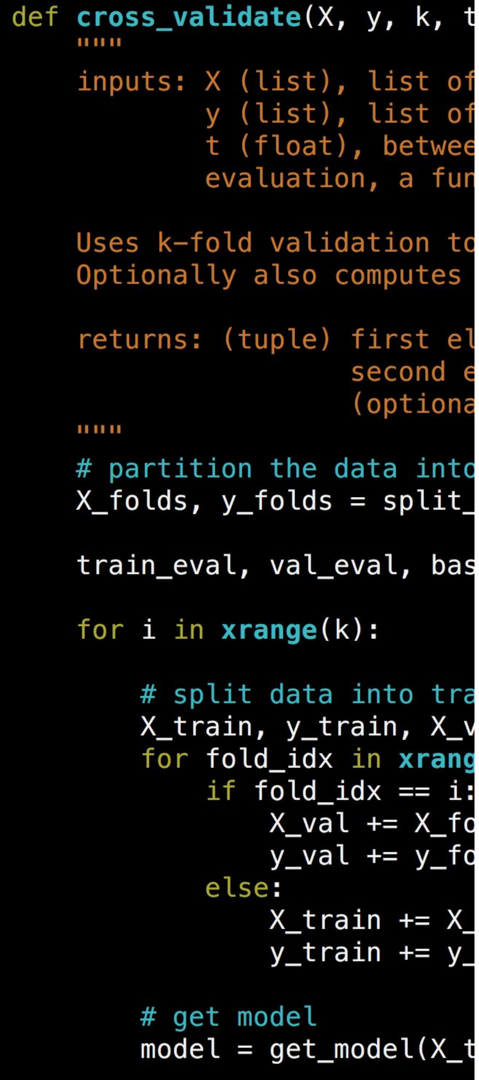


```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

## Coding in Python: the very basics

**Before we begin...**

- pay attention to syntax -- computers are picky!
- pay attention to meaning -- computers are dumb!
- don’t be afraid to run the code you write, even though you think it will fail -- errors are great ways of learning
- remember that you are awesome
- have fun!



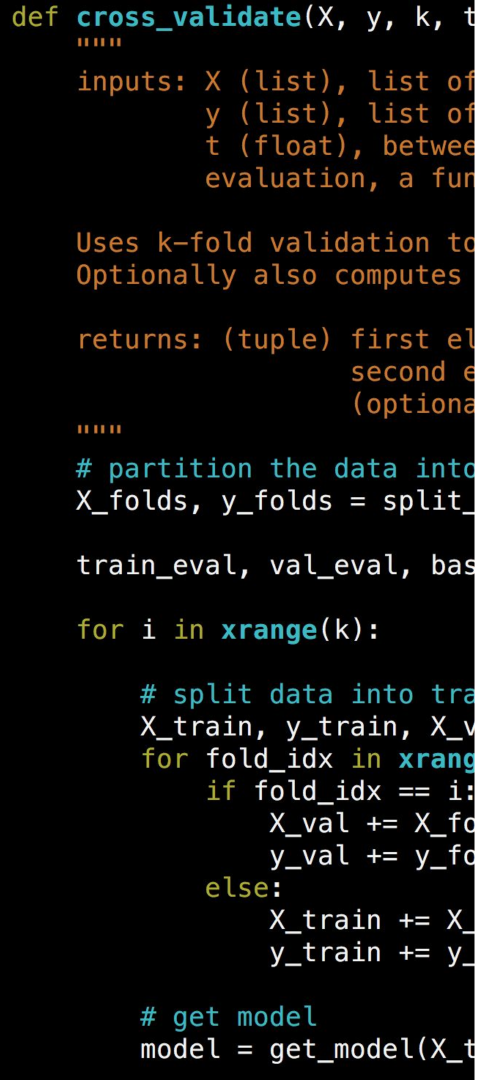
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

# Coding in Python: the very basics

## The Console

- Also called “shell”, “prompt”, etc.
- “Realtime” coding
- Generally not used for serious computation

```
Orhans-MBP:~ orhan$ python
Python 2.7.11 |Anaconda 4.0.0 (x86_64)| (default, Dec  6 2015, 18:57:58)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

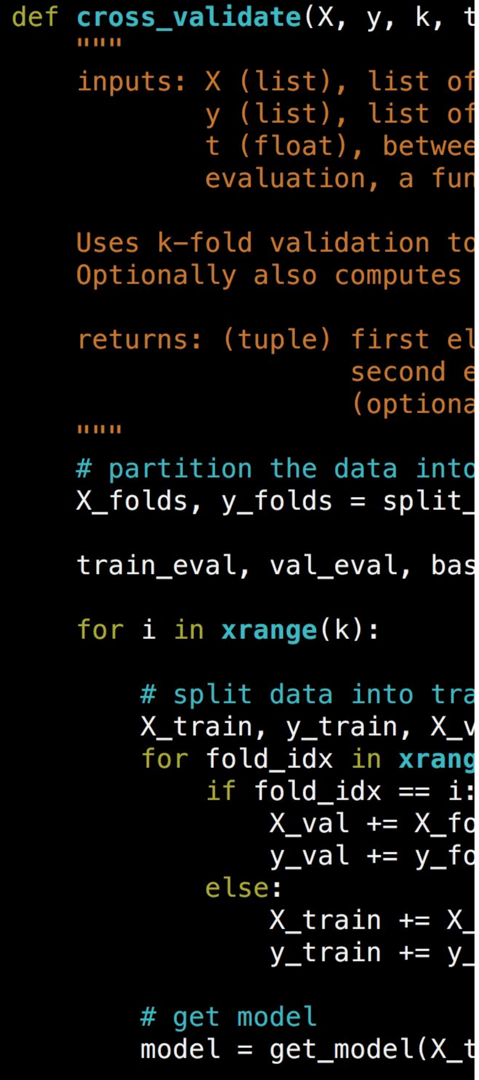
# Coding in Python: the very basics

## Mathematical operations

- Usually work as expected

```
>>> 4 + 2
6
>>> 6 * 10
60
>>> 1 - 3
-2
>>> 10 ** 2
100
```

Power operator: $10^2$



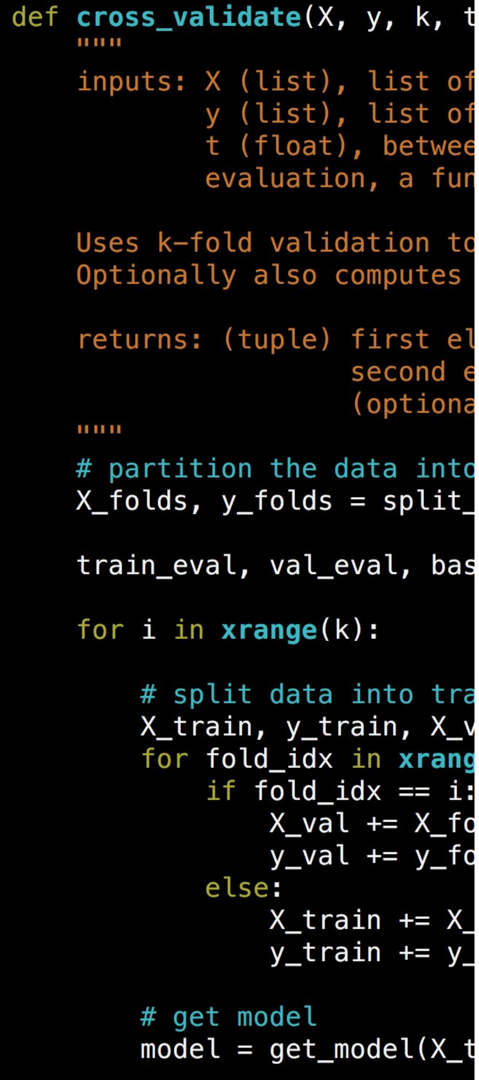
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

# Coding in Python: the very basics

## Mathematical operations

- You can use % for **modulo** operation

```
>>> 5 % 2
1
>>> 10 % 3
1
>>> 1094328971 % 4
3
```



```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

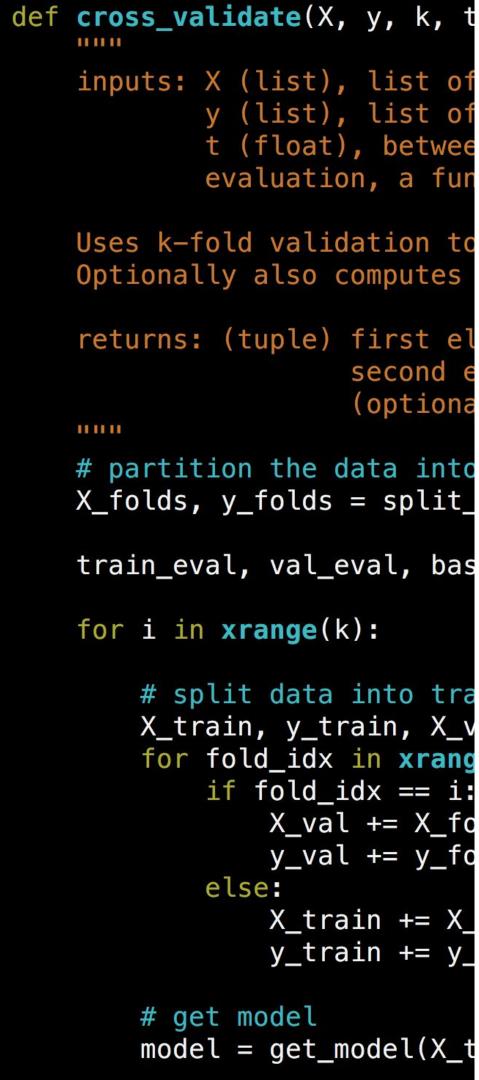
# Coding in Python: the very basics

## Mathematical operations

- The equal sign (=) works differently!

```
>>>> 4 + 2 = 6
  File "<stdin>", line 1
SyntaxError: can't assign to operator

>>> 6 = 6
  File "<stdin>", line 1
SyntaxError: can't assign to literal
```

what



```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

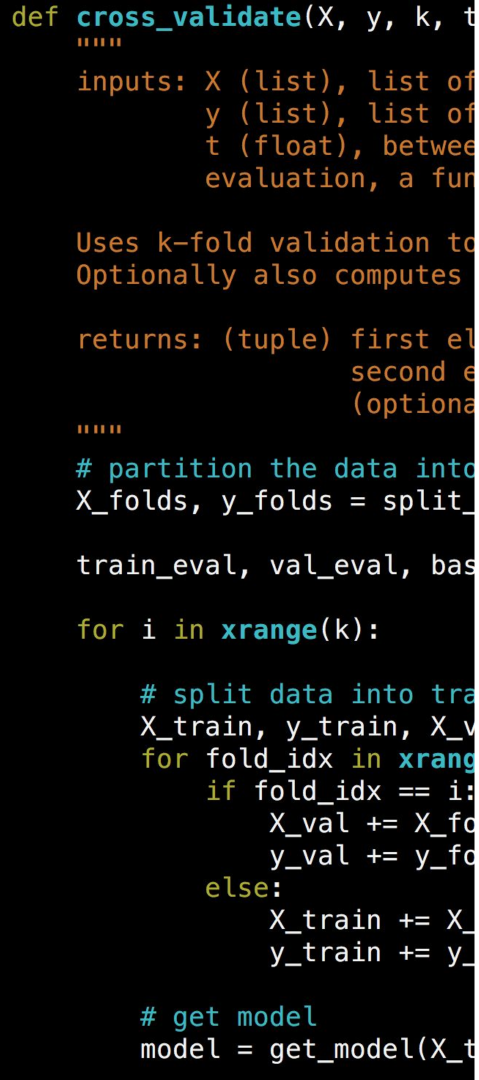
# Coding in Python: the very basics

## Mathematical operations

- The equal sign (=) works differently!

```
>>>> 4 + 2 = 6
  File "<stdin>", line 1
SyntaxError: can't assign to operator

>>> 6 = 6
  File "<stdin>", line 1
SyntaxError: can't assign to literal
```

what



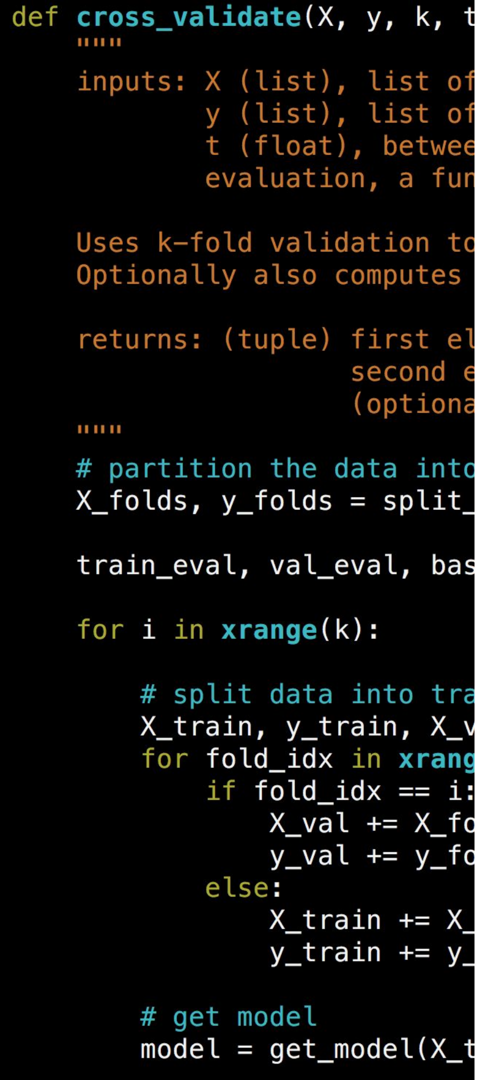
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

## Coding in Python: the very basics

### Mathematical operations

- The equal sign (=) is an *assignment operator*
- “Assign <value of right side> to <value of left side>”
- Right side is always unchanged!

```
>>> a = 5
>>> b = 10
>>> a * b
50
```



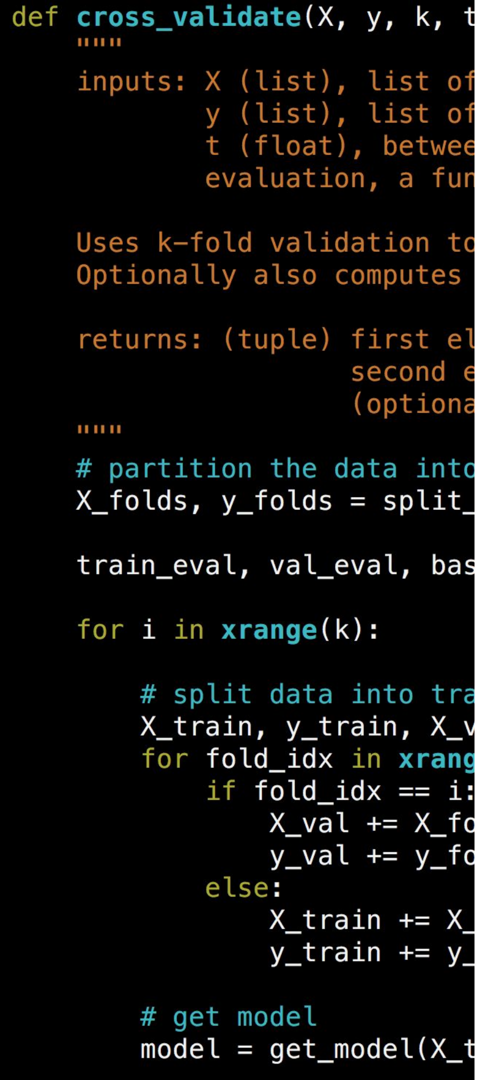
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

## Coding in Python: the very basics

### Mathematical operations

- The equal sign (=) is an *assignment operator*
- “Assign <value of right side> to <value of left side>”
- Right side is always unchanged!

```
>>> a = 5    ——→    value of a is now 5
>>> b = 10   ——→    value of b is now 10
>>> a * b
50
```



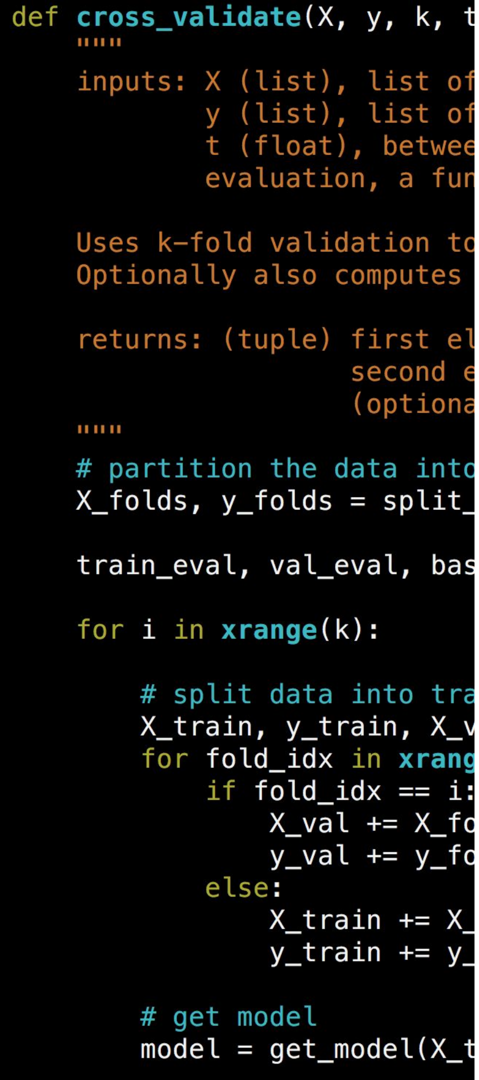
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

# Coding in Python: the very basics

## Assignment operator allows symbolic math

- We can assign values to symbols like “a” and “b”
- We can *also* assign these symbols to each other

```
>>> a = 5       ⟶   value of a is now 5
>>> b = 10      ⟶   value of b is now 10
>>> c = b       ⟶   value of c is now value of b, which is…?
>>> c
10
```



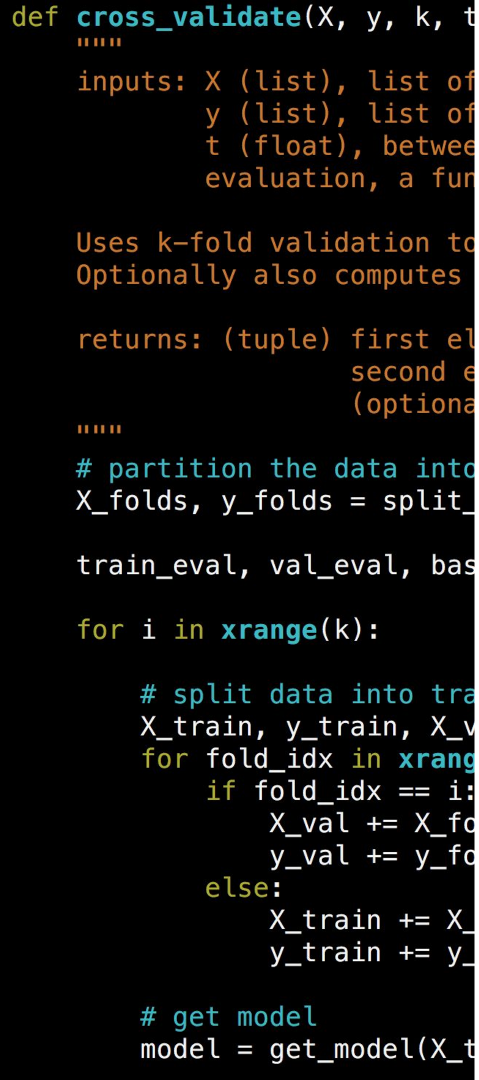
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

## Coding in Python: the very basics

### More on symbols

- You have to define a symbol before using it...
- ... otherwise Python gently warns you

```
>>> a * b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```



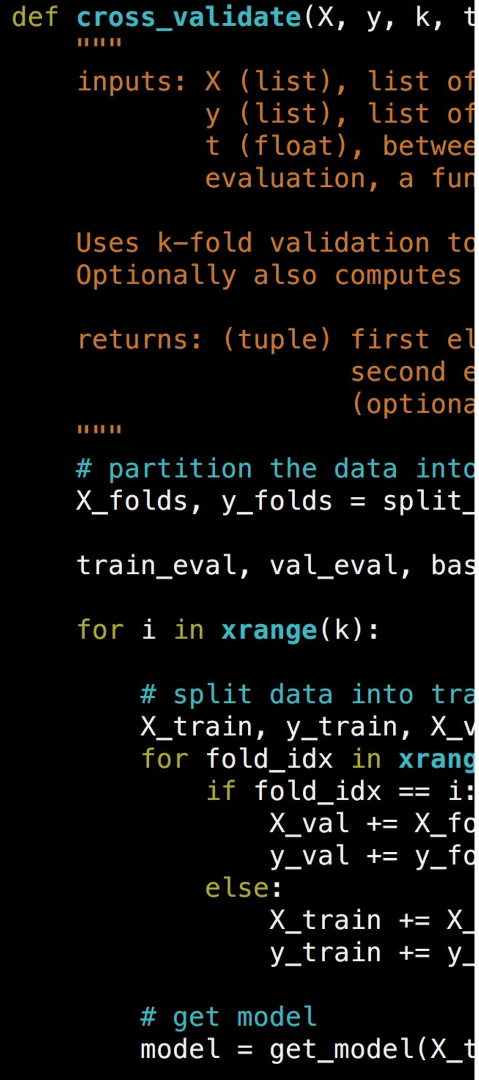
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

## Coding in Python: the very basics

### More on symbols

- “Symbols” are more generally known as “variables”
- They can be named any way you like -- good naming is important!

```
>>> a = 5
>>> b = 9
>>> c = 16
>>> d = (a + b + c) / 3
>>> d
10
```



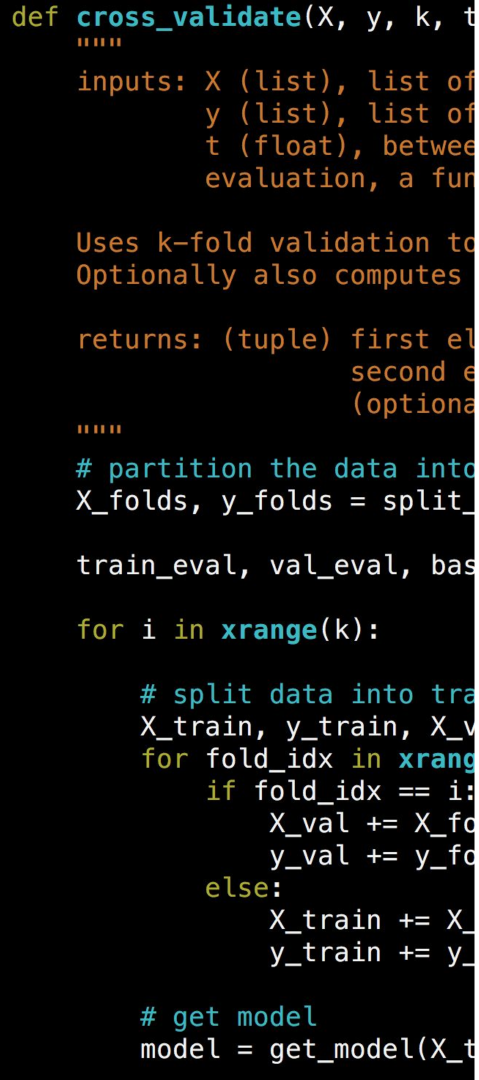
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

# Coding in Python: the very basics

## More on <u>variables</u>

- “Symbols” are more generally known as “variables”
- They can be named any way you like -- good naming is important!

```
>>> a = 5
>>> b = 9
>>> c = 16
>>> d = (a + b + c) / 3
>>> d
10
```



```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

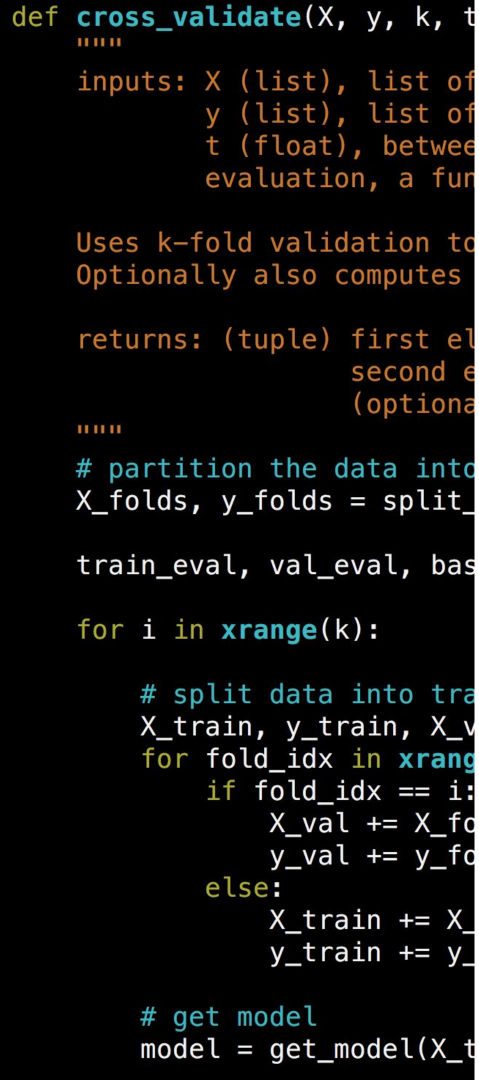
# Coding in Python: the very basics

## More on variables

- "Symbols" are more generally known as "variables"
- They can be named any way you like -- good naming is important!

```
>>> a = 5
>>> b = 9
>>> c = 16
>>> d = (a + b + c) / 3
>>> d
10
```

*Quick note*: you can force order of operations, just like you do in your math class, by using parentheses



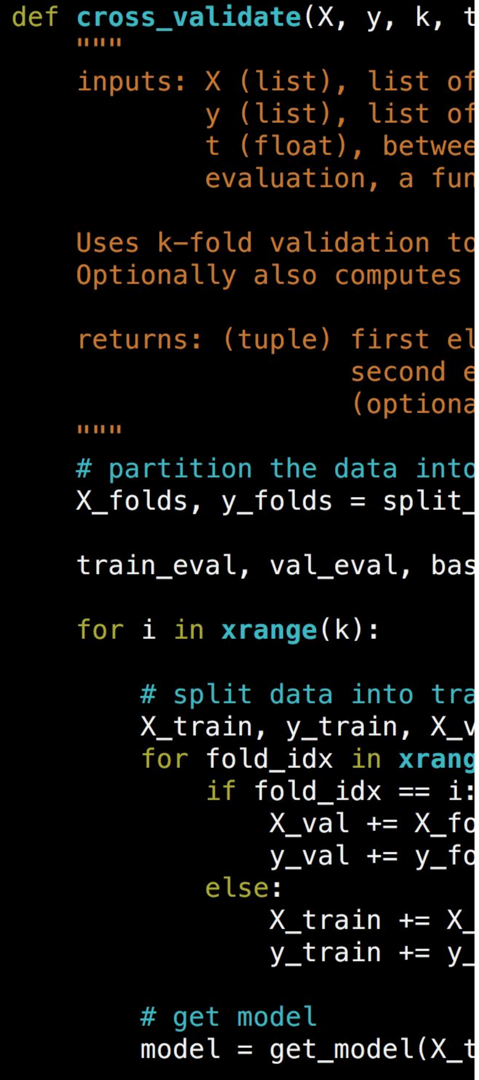
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

# Coding in Python: the very basics

## More on variables

- “Symbols” are more generally known as “variables”
- They can be named any way you like -- good naming is important!

```
>>> age_joe = 5
>>> age_mary = 9
>>> age_lisa = 16
>>> average_age = (age_joe + age_mary + age_lisa) / 3
>>> average_age
10
```



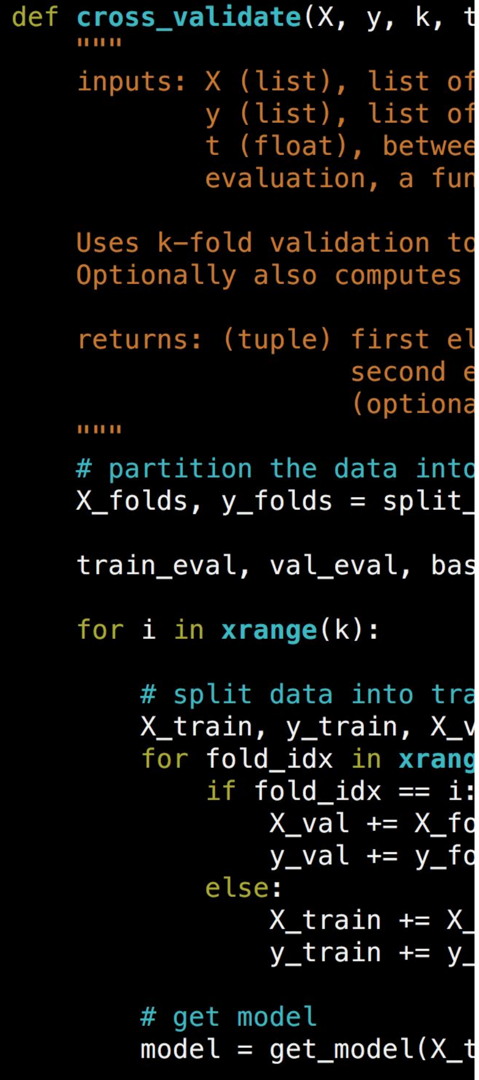
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

# Coding in Python: the very basics

## More on variables

- You can update variables based on their previous values
- Right-hand side is calculated before assigning

```
>>> my_money = 100
>>> my_money = my_money + 5
>>> my_money
105
```



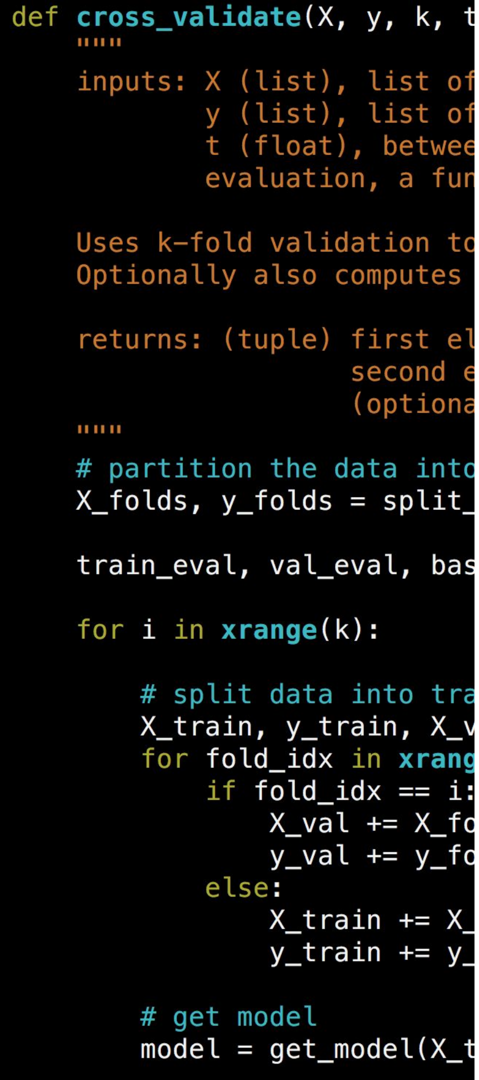
```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

# Coding in Python: the very basics

## More on variables

- You can update variables based on their previous values
- Right-hand side is calculated before assigning

```
>>> my_money = 1000000
>>> my_money = my_money * 1.5
>>> my_money
1500000
```



```
def cross_validate(X, y, k, t
    """
    inputs: X (list), list of
            y (list), list of
            t (float), betwee
            evaluation, a fun

    Uses k-fold validation to
    Optionally also computes

    returns: (tuple) first el
                     second e
                     (optiona
    """
    # partition the data into
    X_folds, y_folds = split_

    train_eval, val_eval, bas

    for i in xrange(k):

        # split data into tra
        X_train, y_train, X_v
        for fold_idx in xrang
            if fold_idx == i:
                X_val += X_fo
                y_val += y_fo
            else:
                X_train += X_
                y_train += y_

        # get model
        model = get_model(X_t
```

## Coding in Python: the very basics

### Printing

- In programming, "print" usually means "display on screen"
- This may seem redundant for now, but it will make sense when we move on to writing longer programs

```
>>> a = 5; b = 10; c = 2
>>> print a * b * c
100
>>> print "hello"
hello
```

## Exercise

What is the final value of b? (printed results not shown)

```
>>> a = 10
>>> b = 2
>>> c = 59
>>> a * b
>>> b * c
>>> a = b
>>> b = c * a
>>> c = c * a
>>> c = c * c
>>> b = a * a
```

# Exercise

What is the final value of b? (printed results not shown)

```
>>> a = 10
>>> b = 2
>>> c = 59
>>> a * b
>>> b * c
>>> a = b
>>> b = c * a
>>> c = c * a
>>> c = c * c
>>> b = a * a
>>> b
4
```

## Exercise

What is the final value of b? (printed results not shown)

```
>>> a = 10      | a is 10     b is undef.  c is undef.
>>> b = 2       | a is 10     b is 2       c is undef.
>>> c = 59      | a is 10     b is 2       c is 59
>>> a * b       | a is 10     b is 2       c is 59
>>> b * c       | a is 10     b is 2       c is 59
>>> a = b       | a is 2      b is 2       c is 59
>>> b = c * a   | a is 2      b is 118     c is 59
>>> c = c * a   | a is 2      b is 118     c is 118
>>> c = c * c   | a is 2      b is 118     c is 13924
>>> b = a * a   | a is 2      b is 4       c is 13924
>>> b
4
```

## Exercise

What is the final value of b? (printed results not shown)

```
>>> a = 10      | a is 10     b is undef.  c is undef.
>>> b = 2       | a is 10     b is 2       c is undef.
>>> c = 59      | a is 10     b is 2       c is 59
>>> a * b       | a is 10     b is 2       c is 59
>>> b * c       | a is 10     b is 2       c is 59
>>> a = b       | a is 2      b is 2       c is 59
>>> b = c * a   | a is 2      b is 118     c is 59
>>> c = c * a   | a is 2      b is 118     c is 118
>>> c = c * c   | a is 2      b is 118     c is 13924
>>> b = a * a   | a is 2      b is 4       c is 13924
>>> b
4
```

## Loops

- Loops allow you to repeat a set of instructions easily

```
>>> a = 0
>>> for i in range(10): ──────────> repeat 10 times
...     a = a + 1
...
>>> a
10
```

## Loops

- Loops allow you to repeat a set of instructions easily

```
>>> for i in range(6): ──────────→ repeat 6 times
...     print "work"
...
work
work
work
work
work
work
```

# Loops

- Repeated instructions are specified by *indentation*

```
>>> for i in range(6): ——→ repeat 6 times
... ____print “work”
...
work
work
work
work
work
work
```

Traditionally 4 spaces

## Loops

- Variable `i` keeps track of iteration number

```
>>> for i in range(6):  ————————→  repeat 6 times
...     print i
...
0
1
2
3
4
5
```

## Loops

- You can nest loops

```
>>> for i in range(2):
...     print "this is the first loop"
...     for j in range(2):
...         print "this is the second loop"
...
```

## Loops

- You can nest loops

```
>>> for i in range(2):
...     print "this is the first loop"
...     for j in range(2):
...         print "this is the second loop"
...
this is the first loop
this is the second loop
this is the second loop
this is the first loop
this is the second loop
this is the second loop
```

# Loops

- You can nest loops

```
>>> for i in range(2):
...     print "this is the first loop"
...     for j in range(2):
...         print "this is the second loop"
...
this is the first loop
this is the second loop
this is the second loop
this is the first loop
this is the second loop
this is the second loop
```

run by first loop

run by second loop

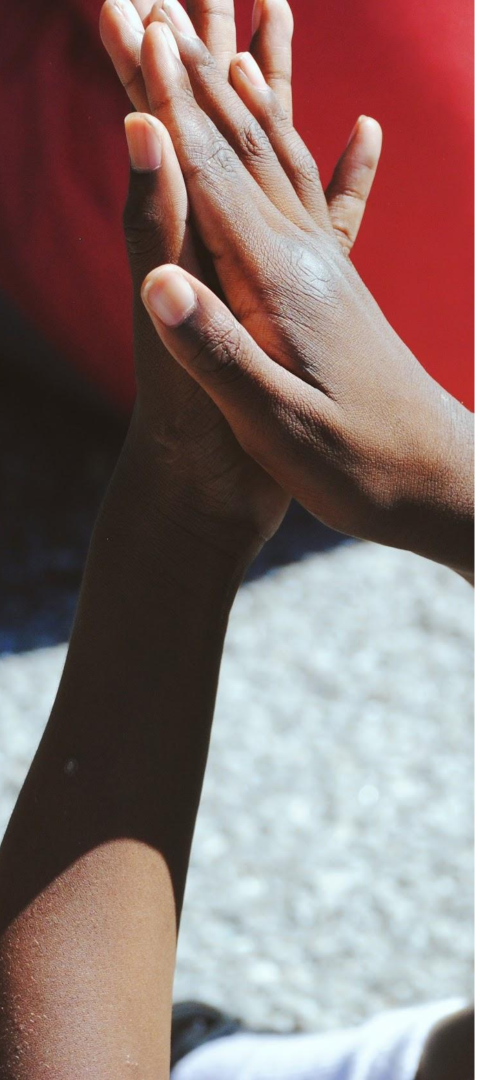
## Loops

- You can nest loops

```
>>> for i in range(2):
...     print "this is the first loop"
...     for j in range(2):
...         print "this is the second loop"
...
this is the first loop
this is the second loop
this is the second loop
this is the first loop
this is the second loop
this is the second loop
```

run by first loop executes 2 times

run by second loop executes 2x2 times

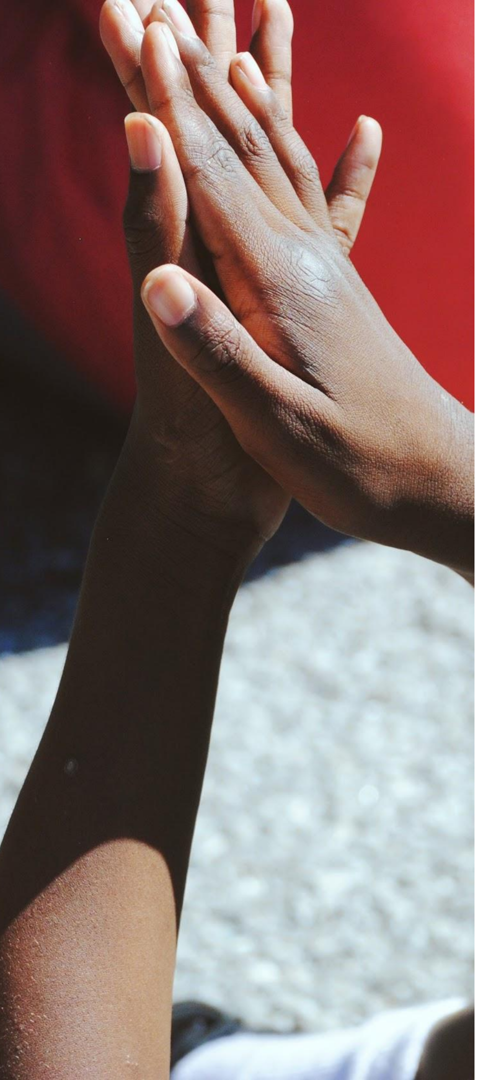


## Conditionals

“if you’re happy and you know it, …”

- A way of introducing logic into your code

```
>>> if 6 > 3:
...     print "Hi!"
...
Hi!
>>> if 3 > 6:
...     print "Hello!"
...
>>>
```



## Conditionals

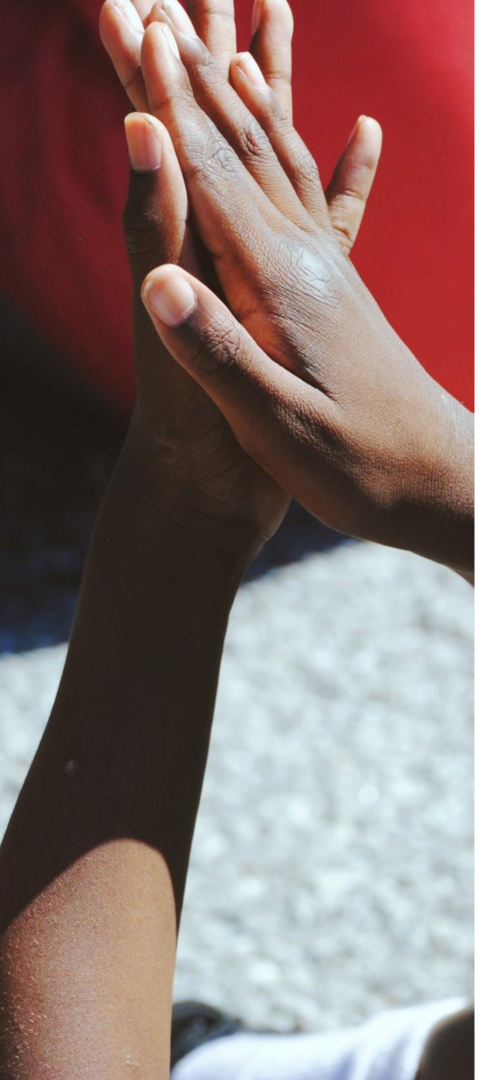
“if you’re happy and you know it, …”

- A way of introducing logic into your code

```
>>> if 6 > 3:
...     print "Hi!"
...
Hi!
>>> if 3 > 6:
...     print "Hello!"
...
>>>
```

$\xrightarrow{}$ printed

$\xrightarrow{}$ NOT printed



## Conditionals

“if you’re happy and you know it, …”

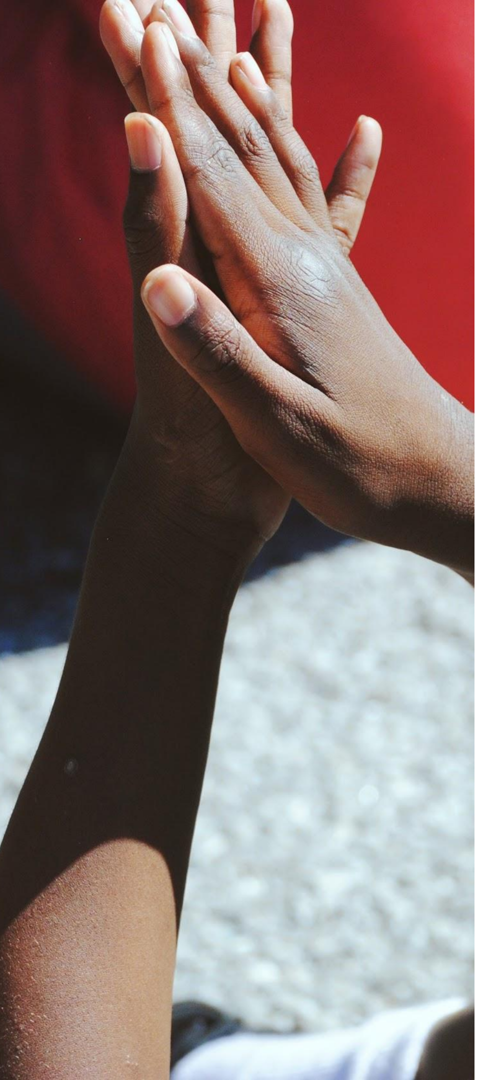
- A way of introducing logic into your code

```
>>> if 6 > 3:
...     print "Hi!"
...
Hi!
>>> if 3 > 6:
...     print "Hello!"
...
>>>
```

printed

NOT printed

Same indentation rules apply



## Conditionals

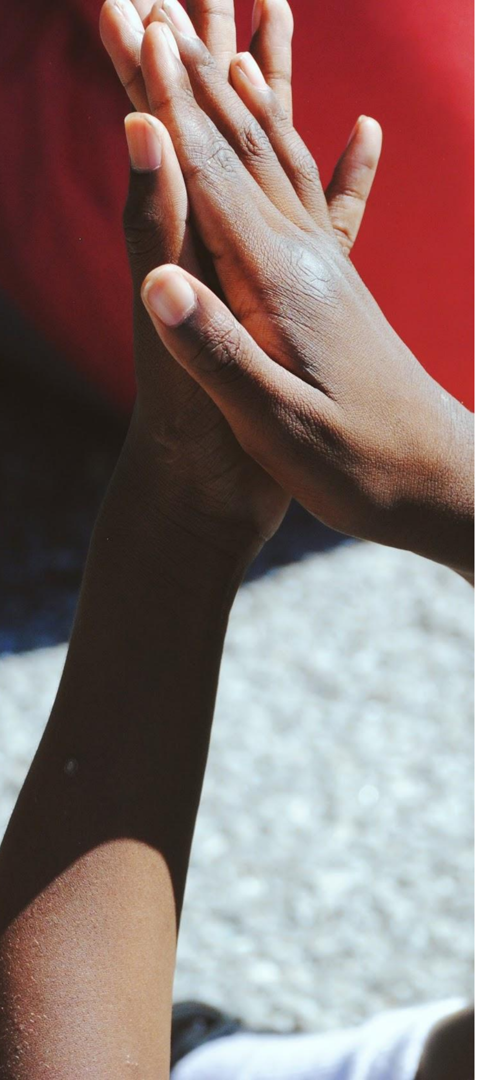
“if you’re happy and you know it, …”

- There is a shorter way of doing this.

```
>>> if 6 > 3:
...       print "Hi!"
...
Hi!
>>> if 3 > 6:
...       print "Hello!"
...
>>>
```

this only runs if this is not true

```
if 6 > 3:
    print “Hi!”
else:
    print “Hello!”
```



## Conditionals: Example

“if you’re happy and you know it, …”

```
>>> current_temp = 75
>>> if current_temp > 80:
...     print "It's pretty hot out there!"
... elif current_temp > 70:
...     print "It's pretty nice now."
... elif current_temp > 60:
...     print "It's still acceptable, I guess?"
... else:
...     print "It's kind of cold out there..."
...
It’s pretty nice now.
```

## Exercise

What does the following piece of code do? (No need to write the output.)

```
>>> for i in range(100):
...     if i % 2 is 0:
...         print "red"
...     if i % 2 is 1:
...         print "blue"
...
```

## Exercise

What does the following piece of code do? (No need to write the output.)

```
>>> for i in range(100):
...     if i % 2 is 0:
...         print "red"
...     if i % 2 is 1:
...         print "blue"
...
red
blue
red
blue
red
... and so on
```

## Live coding!

We'll implement a programming interview classic: FizzBuzz.

For each number from 0 through 99, print ONLY ONE of the following on screen:

- "Fizz" if the number is divisible by 3,
- "Buzz" if the number is divisible by 5,
- "FizzBuzz" if the number is divisible by 15,
- the number itself otherwise.

# We're done!

See you next week!