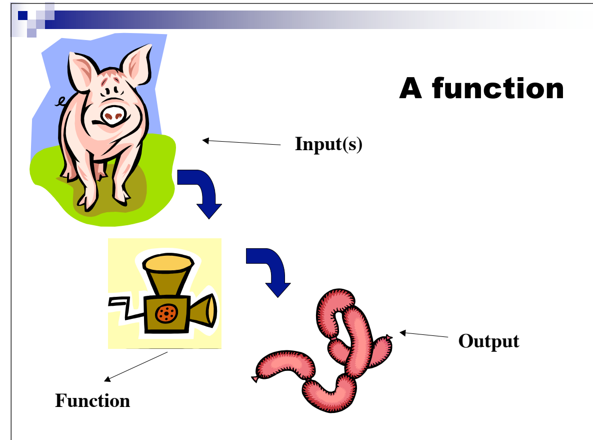
CS1315: Introduction to Media Computation

Making sense of functions

A function



One and only one thing

- We write functions as we do to make them *general* and *reusable*
  - **Programmers hate to have to rewrite something they've written before**
  - **They write functions in a general way so that they can be used in many circumstances.**
- What makes a function *general* and thus *reusable*?
  - **A reusable function does *One and Only One Thing***

Like a good manager, it doesn't micromanage. It delegates details

Compare these two programs

```
def makeSunset(picture):
  for p in getPixels(picture):
    value=getBlue(p)
    setBlue(p, value*0.7)
    value=getGreen(p)
    setGreen(p, value*0.7)
```

Yes, they do exactly the same thing!

makeSunset(somepict) has the same effect in both cases

```
def makeSunset(picture):
  reduceBlue(picture)
  reduceGreen(picture)

def reduceBlue(picture):
  for p in getPixels(picture):
    value=getBlue(p)
    setBlue(p, value*0.7)

def reduceGreen(picture):
  for p in getPixels(picture):
    value = getGreen(p)
    setGreen(p, value*0.7)
```

But this one delegates details

Inputs ("parameters") are placeholders

- When we type in the Command Area

>>> makeSunset(picture)

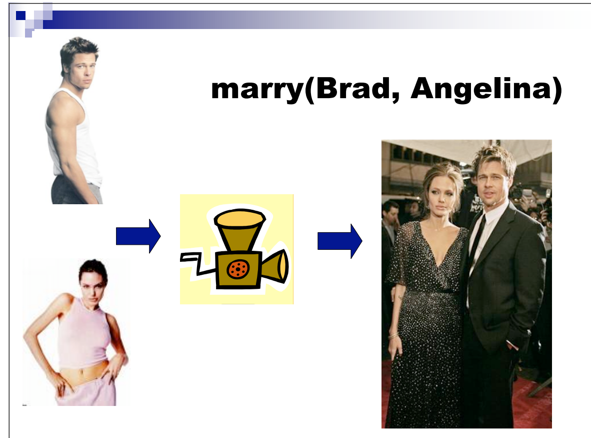
Whatever object that is in the *Command Area* variable **picture** becomes the value of the *placeholder (input) variable* **picture** in

```
def makeSunset(picture):
    reduceBlue(picture)
    reduceGreen(picture)
```

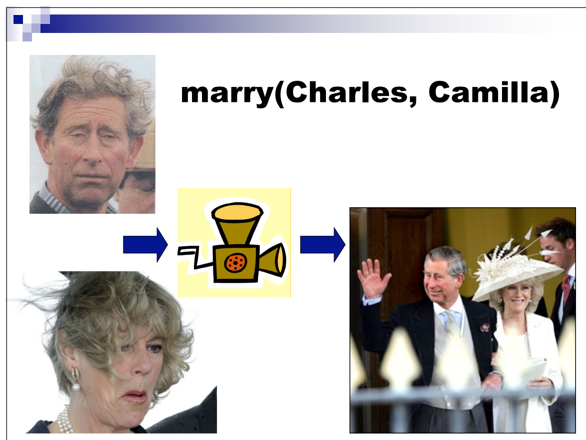
makeSunset's picture is then passed as input to **reduceBlue** and **reduceGreen**, but their input variables are completely different from **makeSunset**'s picture.

- **For the life of the functions, they are the same *values (picture objects)***

marry(Brad, Angelina)



marry(Charles, Camilla)



Input variables as placeholders (example)

- Imagine we have a wedding computer

```
def marry(husband, wife):
    sayVows(husband)
    sayVows(wife)
    pronounce(husband, wife)
    kiss(husband, wife)
```

```
def sayVows(speaker):
    print "I, " + speaker + " blah blah"

def pronounce(man, woman):
    print man + " and " + woman
    print "I now pronounce you…"
    print "You may now kiss the bride"

def kiss(p1, p2):
    print "x x x"
```

So, how do we marry Brad and Angelina?

Input variables as placeholders (example)

- Imagine we have a wedding computer

```
def sayVows(speaker):
    print "I, " + speaker + " blah blah"

def pronounce(man, woman):
    print "I now pronounce you..."

def kiss(p1, p2):
    print "x x x"

def marry(husband, wife):
    sayVows(husband)
    sayVows(wife)
    pronounce(husband, wife)
    kiss(husband, wife)
```


Input variables as placeholders (example)

- Imagine we have a wedding computer

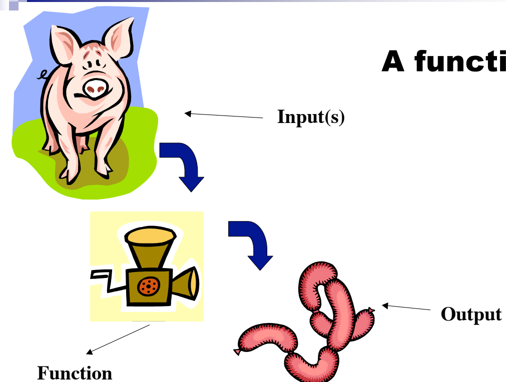
```
def sayVows(speaker):
    print "I, " + speaker + " blah blah"

def pronounce(man, woman):
    print "I now pronounce you..."

def kiss(p1, p2):
    print "x x x"

def marry(husband, wife):
    sayVows(husband)
    sayVows(wife)
    pronounce(husband, wife)
    kiss(husband, wife)
```


Variables within functions *stay* within functions

```
def decreaseRed(picture):
  for p in getPixels(picture):
    value = getRed(p)
    setRed(p, value*0.5)
```

- The variable **value** in **decreaseRed** is created *within* the scope of **decreaseRed**
  - That means that it only exists while decreaseRed is executing
- If we tried to *print value* after running decreaseRed, it would work *ONLY* if we already had a variable defined in the Command Area
  - The name *value* within *decreaseRed* doesn't exist outside of that function
  - We call that a *local* variable

A function



Writing *real* functions

- Functions in the mathematics sense take input and usually return *output.*
  - **Like ord(character) or makePicture(file)**
- What if you create something inside a function that you *do* want to get back to the Command Area?
  - **You can return it.**
  - **We'll talk more about return later—that's how functions *output* something**

Consider these two functions

```
def decreaseRed(picture):
  for p in getPixels(picture):
    value = getRed(p)
    setRed(p, value*0.5)
```

```
def decreaseRed(picture, amount):
  for p in getPixels(picture):
    value = getRed(p)
    setRed(p, value * amount)
```

• First, it's perfectly okay to have *multiple* inputs to a function.

• The new **decreaseRed** now takes an input of the multiplier for the red value.

- **decreaseRed(picture, 0.5)** would do the same thing
- **decreaseRed(picture, 1.25)** would *increase* red 25%