List Recursion Examples & Recursive Procedures

One-Slide Summary

- Recursive functions that operate on lists have a similar structure. **list_cruncher** is a higher-order function that can be used to implement many others.
- Decisions in a function can be abstracted out by adding a function argument. For example, **find_closest_number** is just **find_closest** plus a function defining what a **close_number** is.
- The Fibonacci numbers are a **recursively-defined sequence**.
- Almost all music uses a **stack** structure: starts on the tonic, repeats similar patterns in a structured way, ends on the tonic.

Outline

- map and filter
- list_cruncher
- find_closest_number
  - Reminder: procedure definition strategy!
- find_closest
- Fibonacci numbers
- Recursive Transition Networks
  - vs. Backus-Naur Form Grammars
- Musical Harmony

Map and Filter **Combined**

[x * x for x in [1,2,3,4,5] if is_odd(x)]
???

Map and Filter **Combined**

[x * x for x in [1,2,3,4,5] if is_odd(x)]
[1, 9, 25]

[x/3 for x in [11,22,33,44] if is_odd(x+1)]
???

Map and Filter **Combined**

[x * x for x in [1,2,3,4,5] if is_odd(x)]
[1, 9, 25]

[x/3 for x in [11,22,33,44] if is_odd(x+1)]
[7, 14]

[*transform* for *name* in *list* if *predicate*]
[*map* for *elt* in *list* if *filter*]

Similarities and Differences

```
def map(work, lst):
  if not lst:
    return []
  return [work(lst[0])] + \
         map(work,lst[1:])
```

```
def sumlist(lst):
  if not lst:
    return 0
  return lst[0] + \
    sumlist(lst[1:])
```


Similarities and Differences

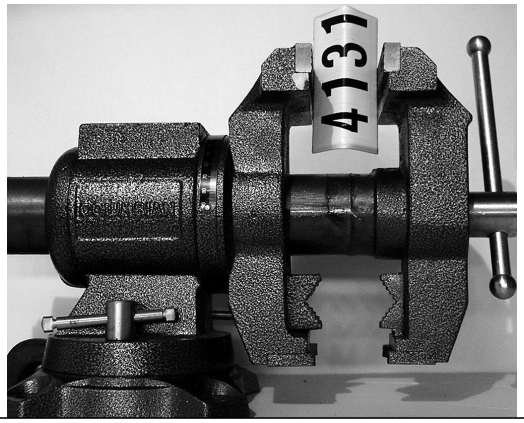
```
def map(work, lst):
  if not lst:
    return []
  return [work(lst[0])] + \
         map(work,lst[1:])
```

```
def sumlist(lst):
  if not lst:
    return 0
  return lst[0] + \
    sumlist(lst[1:])
```

```
def list_cruncher(..., lst):
  if not lst:
    return base_result
  return combine(lst[0], list_cruncher(..., lst[1:]))
```


How could this work?

- I want to crunch all the lists. How would I get started?



One Ring To Rule Them All

```
def list_cruncher(base, proc, combine, lst):
  if not lst:
    return base
  return combine(proc(lst[0]), \
    list_cruncher(base,proc,combine,lst[1:])

def sumlist(lst):
   return list_cruncher(0, lambda (x) : x, \
    lambda y,z : y+z, lst)
```


One Ring To Rule Them All

```
def list_cruncher(base, proc, combine, lst):
  if not lst:
    return base
  return combine(proc(lst[0]), \
    list_cruncher(base,proc,combine,lst[1:])

def map(workfun, lst):
   return list_cruncher([], workfun, \
    lambda y,z : [y]+z, lst)
```


Crunchy Center Challenge

```
def list_cruncher(base, proc, combine, lst):
  if not lst:
    return base
  return combine(proc(lst[0]), \
    list_cruncher(base,proc,combine,lst[1:])

def length(lst):
   if not lst: return 0
   return 1 + length(lst[1:])
```

How can we define length in terms of list_cruncher?

Crunchy Center Challenge

```
def list_cruncher(base, proc, combine, lst):
  if not lst:
    return base
  return combine(proc(lst[0]), \
     list_cruncher(base,proc,combine,lst[1:])

def length(lst):
  return list_cruncher(0, lambda x : 1, \
    lambda y,z: y+z, lst)
```


Python Elegance Corner

```
def mymin(a,b):
  if a < b:
    return a
  else:
    return b

def mymin(a,b):
  return a if a<b else b
```

- These are the same! Both are full credit!

Python Elegance Corner 2

```
>>> (8 if 1 > 5 else 2) + 3
5
>>> foo = lambda x : 0 if x == 5 else x
>>> foo(4)
4
>>> foo(5)
0
>>> foo(6)
6
```


Crunchy Center Rematch

```
def list_cruncher(base, proc, combine, lst):
  if not lst:
    return base
  return combine(proc(lst[0]), \
     list_cruncher(base,proc,combine,lst[1:])

def filter(pred, lst):
   if not lst: return []
   if pred(lst[0]): return [lst[0]] + filter(pred,lst[1:])
   return filter(pred,lst[1:])
```

How can we define filter in terms of list_cruncher?
Hint: [x] if pred(x) else [ ]

list_cruncher crunches filter!

```
def list_cruncher(base, proc, combine, lst):
  if not lst:
    return base
  return combine(proc(lst[0]), \
     list_cruncher(base,proc,combine,lst[1:])

def filter(pred, lst):
  return list_cruncher([], lambda x : \
   [x] if pred(x) else [], lambda y,z: y+z, lst)
```


Liberal Arts Trivia: Drama

- In this 1948 play by Samuel Beckett has been called “the most significant English-language play of the 20th century”. The minimal setting calls to mind “the idea of the ‘lieu vague’, a location which should not be particularised”, and the play features two characters who never meet the title character.

Liberal Arts Trivia: History

- At the height of its power, in the 16th and 17th century, this political organization spanned three continents. It controlled much of Southeastern Europe, the Middle East and North Africa, and contained 29 provinces and multiple vassal states. Noted cultural achievements include architecture (vast inner spaces confined by seemingly weightless yet massive domes, harmony between inner and outer spaces, articulated light and shadow, etc.), classical music, and cuisine.

find-closest-number

- The function **find_closest_number** takes two arguments. The first is a single number called the goal. The second is a *non-empty* list of numbers. It returns the number in the input list that is closest to the goal number.

```
>>> find_closest_number(150, [101,110,120,157,340,588])
157
>>> find_closest_number(12, [4,11,23])
11
>>> find_closest_number(12, [95])
95
```

We'll do this one together!

Recall The Strategy!

Be optimistic!

Assume you can define:

find_closest_number(goal, numbers)

that finds the closest number to goal from a small list of numbers.

What if there is *one more number*?

Can you write a function that finds the closest number to match from the first number and the other numbers?

find-closest-number hint

One Approach for the Recursive Case:

You have two possible answers: the current car of the list and the result of the recursive call. Compare them both against the goal number, and return the one that is closer.

Optimistic Function

```
def find_closest_number(goal, numbers):
 # base case missing for now!
 if abs(goal - numbers[0]) < abs(goal - \
    find_closest_number(goal,numbers[1:])):
   return numbers[0]
 else:
   return find_closest_number(goal, numbers[1:])
```


Defining Recursive Procedures

2. Think of the simplest version of the problem (almost always [ ]), something you can already solve. (**base case**)

Is [ ] the base case for find_closest_number?

find_closest_number defined!

```
def find_closest_number(goal, numbers):
  if len(numbers) == 1:      # base case
    return numbers[0]        # return the only element
  if abs(goal - numbers[0]) < abs(goal - \
      find_closest_number(goal,numbers[1:])):
    return numbers[0]
  else:
    return find_closest_number(goal, numbers[1:])
```

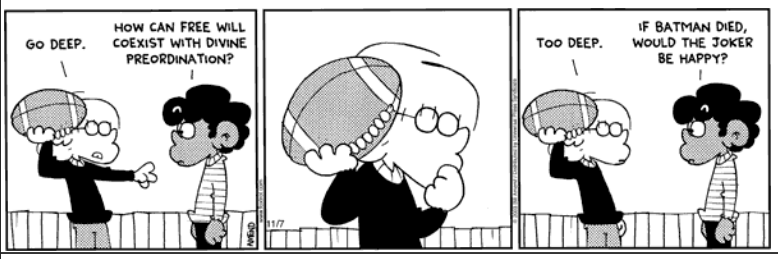

Python Interactions

```
>>> def find_closest_number(goal, numbers):
    if len(numbers) == 1:   # base case
        return numbers[0]   # return the only element
    if abs(goal - numbers[0]) < abs(goal -\
                                find_closest_number(goal,numbers[1:])):
        return numbers[0]
    else:
        return find_closest_number(goal, numbers[1:])


>>> find_closest_number(150, [101, 110, 120, 157, 340, 588])
157
>>> find_closest_number(0, [1])
1
>>> find_closest_number(0, [])
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "<console>", line 4, in find_closest_number
IndexError: list index out of range
```

Generalizing find-closest-number

- How would we implement find_closest_number_without_going_over?
- What about find_closest_word?
- …



Generalizing find-closest-number

- How would we implement find_closest_number_without_going_over?
- What about find_closest_word?
- **The "closeness" metric should be a procedure we pass in!**

find_closest defined!

```
def find_closest(goal, lst, closeness):
  if len(lst) == 1:      # base case
    return lst[0]        # return the only element
  if closeness(goal, lst[0]) < closeness(goal, \
      find_closest(goal, lst[1:], closeness)):
    return lst[0]
  else:
    return find_closest(goal, lst[1:], closeness)
```

How can we implement find_closest_number using find_closest?

Using find_closest

```
def find_closest_number(goal, numbers):
  return find_closest(goal, numbers, \
    lambda a, b : abs(a-b) )

def find_closest_below(goal, numbers):
  return find_closest(goal, numbers, \
    lambda a, b: a-b if a >= b else maxint)
```

Duplicate Work?

```
def find_closest(goal, lst, closeness):
  if len(lst) == 1:       # base case
    return lst[0]         # return the only element
  if closeness(goal, lst[0]) < closeness(goal, \
      find_closest(goal, lst[1:], closeness)):
    return lst[0]
  else:
    return find_closest(goal, lst[1:], closeness)
```

How can we avoid evaluating **find_closest** twice?

Helper Procedures

```
def pick_closer(a, b, closeness):
  return a if closeness(a) < closeness(b) else b

def find_closest(goal, lst, closeness):
  if len(lst) == 1:       # base case
    return lst[0]         # return the only element
  return pick_closer(lst[0], \
    find_closest(goal, lst[1:], closeness), \
    closeness)
```

Where have we seen something like this before?

Photomosaics!

```
def pick_closer(a, b, closeness):
  return a if closeness(a) < closeness(b) else b

def find_closest(goal, lst, closeness):
  if len(lst) == 1:       # base case
    return lst[0]         # return the only element
  return pick_closer(lst[0], \
    find_closest(goal, lst[1:], closeness), \
    closeness)
```

find_best_match from PS1 is just find_closest using closer_color as closeness!

Liberal Arts Trivia: Philosophy

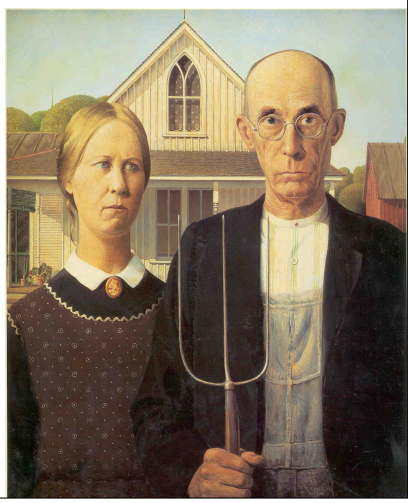
- This branch of philosophy, which Aristotle called “First Philosophy”, investigates principles of reality transcending those of any particular science. It is concerned with explaining the ultimate nature of being and the world (e.g., determinism and free will, mind and matter, space and time). Its modern name comes from the fact that Aristotle's chapters about it were placed “beyond” his chapters on matter and force.

Liberal Arts Trivia: Film Studies

- Born in 1965 to Muslim parents, this Indian actor has starred in flims such as *Kuch Kuch Hota Hai*, *Kal Ho Naa Ho*, *Veer-Zaara*, and *Devdas*. In 2008, *Newsweek* named him one of the 50 most powerful people in the world. He has replaced Amitabh “Big B” Bachchan as the host of Kaun Banega Crorepti, and has won India's Padma Shri, a life-sized wax statue at Madame Tussaud's, and the French government's Ordre des Arts et des Lettres.

Liberal Arts Trivia: Painting

- Name this 1930 oil-on-beaverboard painting by Grant Wood. It is one of the most familiar images of 20th century American art and has achieved an iconic status.



GEB Chapter V

Consider the optional-optional reading!

You could spend the rest of your life just studying things in this chapter (25 pages)!

- Music Harmony
- Stacks and Recursion
- Theology
- Language Structure
- Number Sequences
- Chaos
- Fractals (PS3 out today. Start early. Why?)
- Quantum Electrodynamics (later lecture)
- DNA (later lecture)
- Sameness-in-differentness
- Game-playing algorithms (later lecture)

Fibonacci's Problem

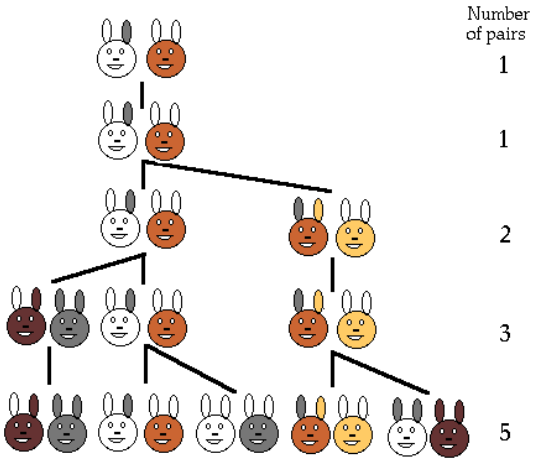
Filius Bonacci, 1202 in Pisa:

Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits.

Suppose that our rabbits **never die** and that the female **always** produces one new pair (one male, one female) **every month** from the second month on.

How many pairs will there be in one year?

Rabbits



From http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fibnat.html

Fibonacci Numbers

GEB p. 136:

These numbers are best defined **recursively** by the pair of formulas

FIBO (*n*) = FIBO (n – 1) + FIBO (n – 2)
for n > 2

FIBO (1) = FIBO (2) = 1
for n <= 2

Can we turn this into a Python procedure?

Defining FIBO

1. Be optimistic - assume you can solve it, if you could, how would you solve a bigger problem.
2. Think of the simplest version of the problem, something you can already solve.
3. Combine them to solve the problem.

These numbers are best defined recursively by the pair of formulas

FIBO (*n*) =
FIBO (n – 1)
+ FIBO (n – 2)
for n > 2

FIBO (1) = FIBO (2) = 1

Defining fibo

```
# fibo(n) evaluates to the nth Fibonacci
# number
def fibo(n):
  if n == 1 or n == 2:
    return 1 # base case
  return fibo(n-1) + \
    fibo(n-2)
```

FIBO (1) = FIBO (2) = 1

FIBO (*n*) =
FIBO (n – 1)
+ FIBO (n – 2)
for n > 2

Concise fibo

```
# fibo(n) evaluates to the nth Fibonacci
# number
def fibo(n):
  return 1 if n <= 2 else fibo(n-1) + fibo(n-2)
```

FIBO (1) = FIBO (2) = 1

FIBO (*n*) =
  FIBO (n – 1)
  + FIBO (n – 2)
  for n > 2

Fibo Results

```
>>> fibo(2)
1
>>> fibo(3)
2
>>> fibo(4)
3
>>> fibo(10)
55
>>> fibo(60)
```

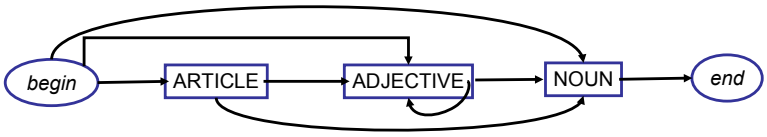
Still working after 4 hours…

Why can't our 4Mx Apollo Guidance Computer figure out how many rabbits there will be in 5 years?

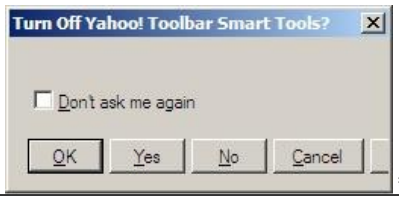
To be continued...

Recursive Transition Networks

ORNATE NOUN

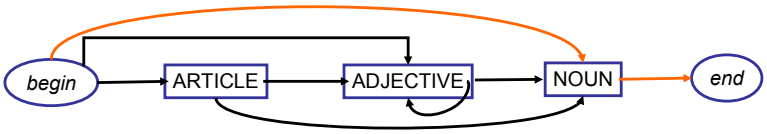


Can we describe this using Backus Naur Form?



Recursive Transition Networks

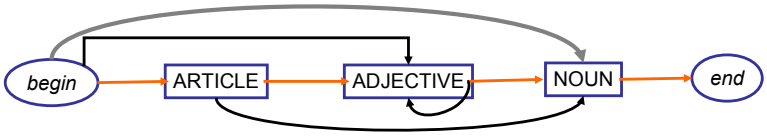
ORNATE NOUN



ORNATE NOUN ::= NOUN

Recursive Transition Networks

ORNATE NOUN

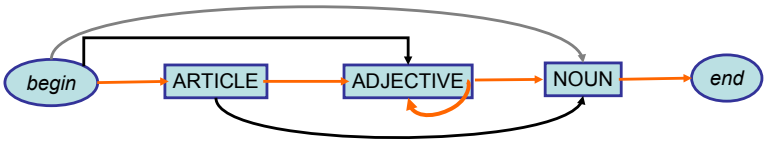


ORNATE NOUN ::= NOUN

ORNATE NOUN ::= ARTICLE ADJECTIVE NOUN

Recursive Transition Networks

ORNATE NOUN



ORNATE NOUN ::= ARTICLE ADJECTIVE NOUN

ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE NOUN

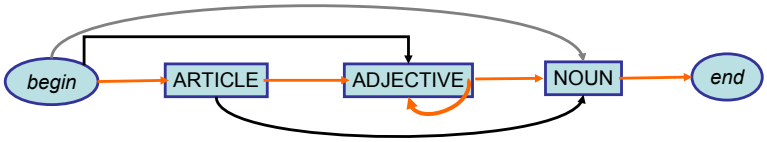
ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE ADJECTIVE NOUN

ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE ADJECTIVE ADJECTIVE NOUN

ORNATE NOUN ::= ARTICLE ADJECTIVE ADJECTIVE ADJECTIVE ADJECTIVE ADJECTIVE NOUN

Recursive Transition Networks

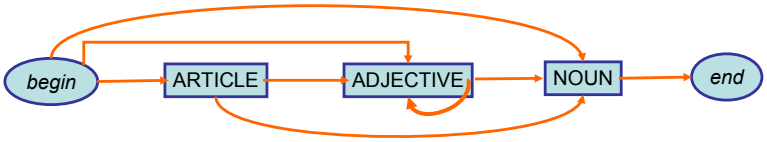
ORNATE NOUN



ORNATE NOUN ::= ARTICLE ADJECTIVES NOUN
ADJECTIVES ::= ADJECTIVE ADJECTIVES
ADJECTIVES ::=

Recursive Transition Networks

ORNATE NOUN



ORNATE NOUN ::= OPTARTICLE ADJECTIVES NOUN
ADJECTIVES ::= ADJECTIVE ADJECTIVES
ADJECTIVES ::= ε
OPTARTICLE ::= ARTICLE
OPTARTICLE ::= ε

Which notation is *better*?

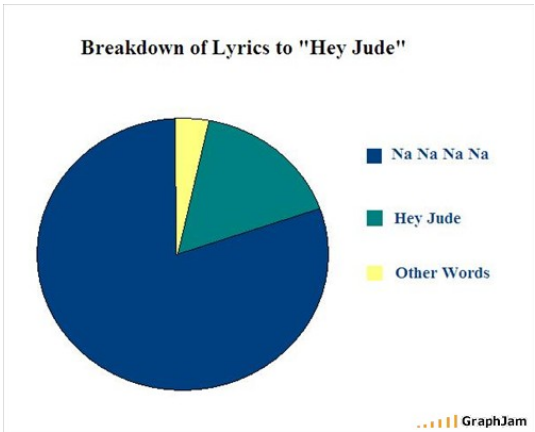
Music Harmony

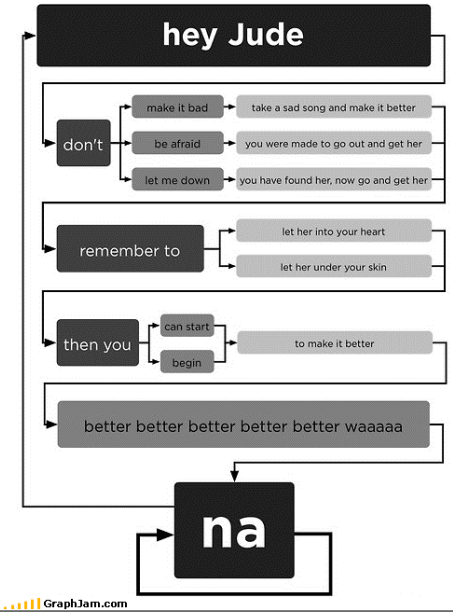
Kleines Harmonisches Labyrinth

(Little Harmonic Labyrinth)

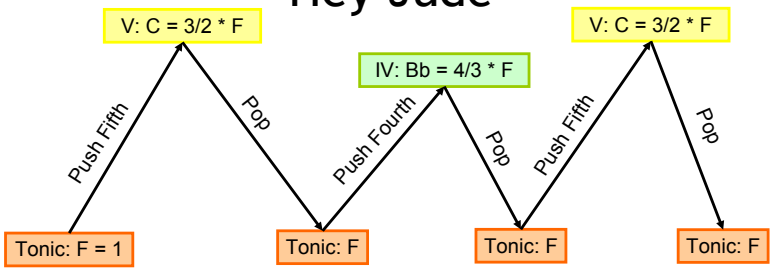
Hey Jude

John Lennon and Paul McCartney, 1968

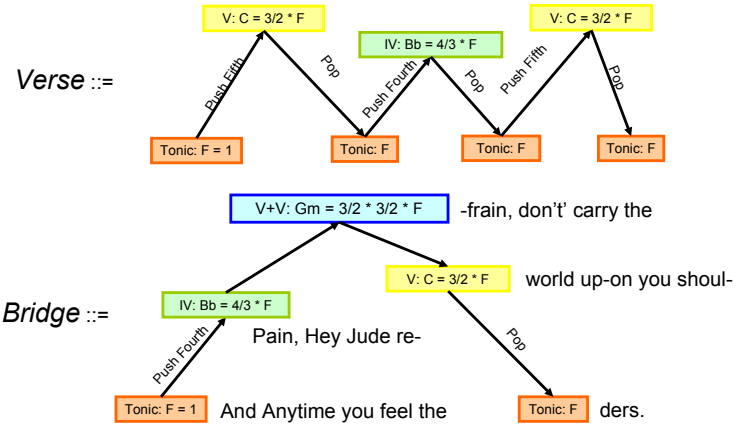




Hey Jude



Tonic: Hey Jude, don't make it
V: bad. take a sad song and make it
Tonic: better Re-
IV: member to let her into your
Tonic: heart, then you can
V: start to make it bet-
Tonic: -ter.



HeyJude ::= *Verse VBBN VBBN Verse Verse Better Coda*
VBBN ::= *Verse Bridge Bridge Nanana (ends on C)*
Coda ::= F Eb Bb F *Coda*

Music

- **Almost All Music Is Like This**
  - Pushes and pops the listener's stack, but doesn't go too far away from it
  - Repeats similar patterns in structured way
  - Keeps coming back to Tonic, and Ends on the Tonic
- Any famous Beatles song that doesn't end on Tonic?

Charge

- **Challenge:** Try to find a "pop" song with a 3-level deep harmonic stack
- **PS3:** due soon!.

Be optimistic!

You know everything you need to finish it now, and it is longer than PS2, so get started now!

Beatles: "A Day in the Life" (starts on G, ends on E)

Homework

- Start Problem Set 3 Now
  - No, really.
  - Due way too soon ...
- PS3 has associated Reading