Lecture 12

August 30, 2016

Clustering, Classifiers, Information Theory, Dynamic Programming

Reminder: start the recording

Naïve Bayes Classification

- Recall Bayes Theorem

$$PP(AA|BB) = \frac{PP(BB|AA)PP(AA)}{PP(BB)}$$

- Language classification

$$PP(llllllll|tttttttt) = \frac{PP(tttttttt|lllllll)PP(lllllll)}{PP(ttttttt)}$$

Naïve Bayes language classifier

$$PP(llllllll|tttttttt) = \frac{PP(tttttttt|llllllll)\cancel{PP(llllllll)}}{\cancel{PP(tttttttt)}}$$

$PP(tttttttt)$ - Prior probability that the text is in (some) language: 1.0

$PP(llllllll)$ – Prior probability of encountering each language: assume all languages are equally likely

Naïve assumption

All features are independent of all others

For this task, a “feature” is the occurrence of a word

$$PP(llllllll\,|tttttttt)$$

$$= PP(lllllll|ww_1, ww_2, \ldots ww_{nn})$$

$$= PP(ww_1, ww_2, \ldots ww_{nn}\,|lllllll)$$

$$= \overset{nn}{\underset{ii=1}{\text{�}}} PP(ww_{ii}|lllllll)$$

$$\text{logprob}(lllllll\,|ttttttt) = \overset{nn}{\underset{ii=1}{\text{�}}} PP(ww_{ii}|lllllll)$$

Last step

$$\text{logprob}\ l(llllll\ tt|tttttt) = \underset{ii=1}{\overset{nn}{\text{�}}} \log PP(ww_{ii}|llllllll)$$

This gives the (log-)probability of a language given a text. To find the **most** probable language:

$$L\text{�} = \text{argmax}_{jj} \underset{ii=1}{\overset{nn}{\text{�}}} \log PP(ww_{ii}|llllllll_{jj})$$

k-Nearest Neighbor Classification

- “Classification by peer pressure”
- Instance-based learning (“lazy learning”)
  - No training
- Need a distance metric
- Test instance is given the same label as its kk closest neighbors
  - Voting schemes resolve conflict
- To test, need to calculate distance to all training instances
  - This can be slow at runtime

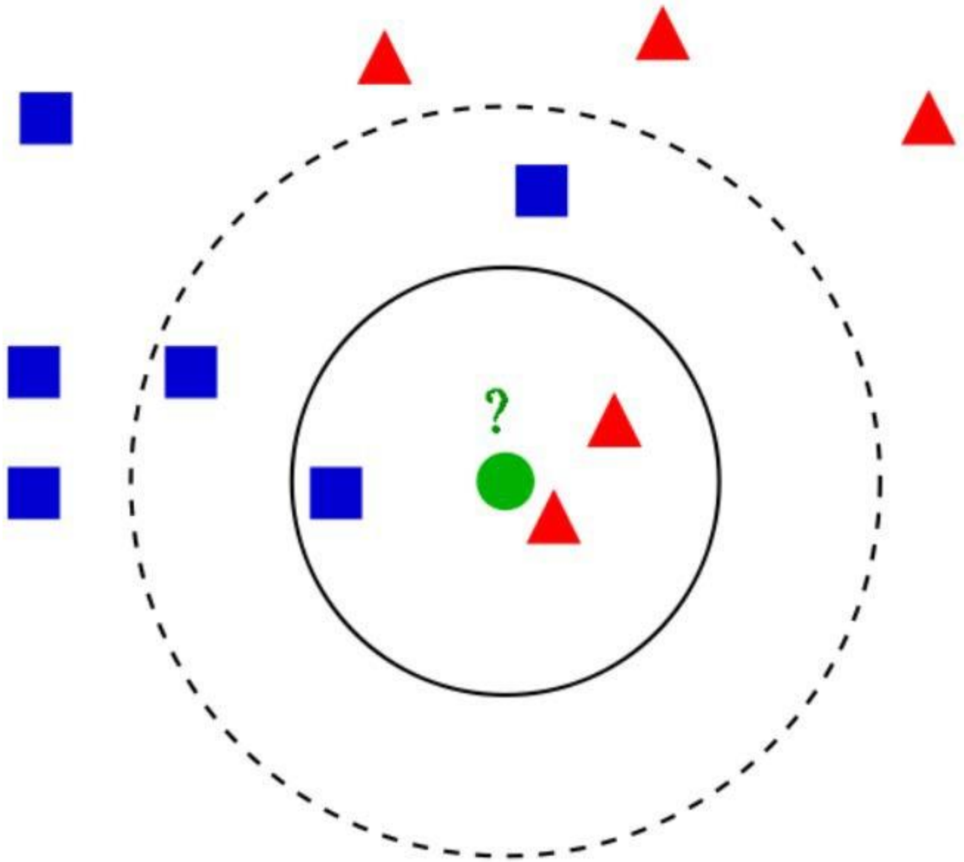
kNN Distance metric

- Should be fast to calculate
- Usually, just a high-dimensional vector space model (VSM)

VSM
Vector Space Model
n-Dimensional Euclidian space
where each feature has its
own axis of variability

SVM
Support Vector Machine
A particular type of quadratic
programming classifier

kNN Classification



http://en.wikipedia.org/wiki/File:KnnClassification.svg

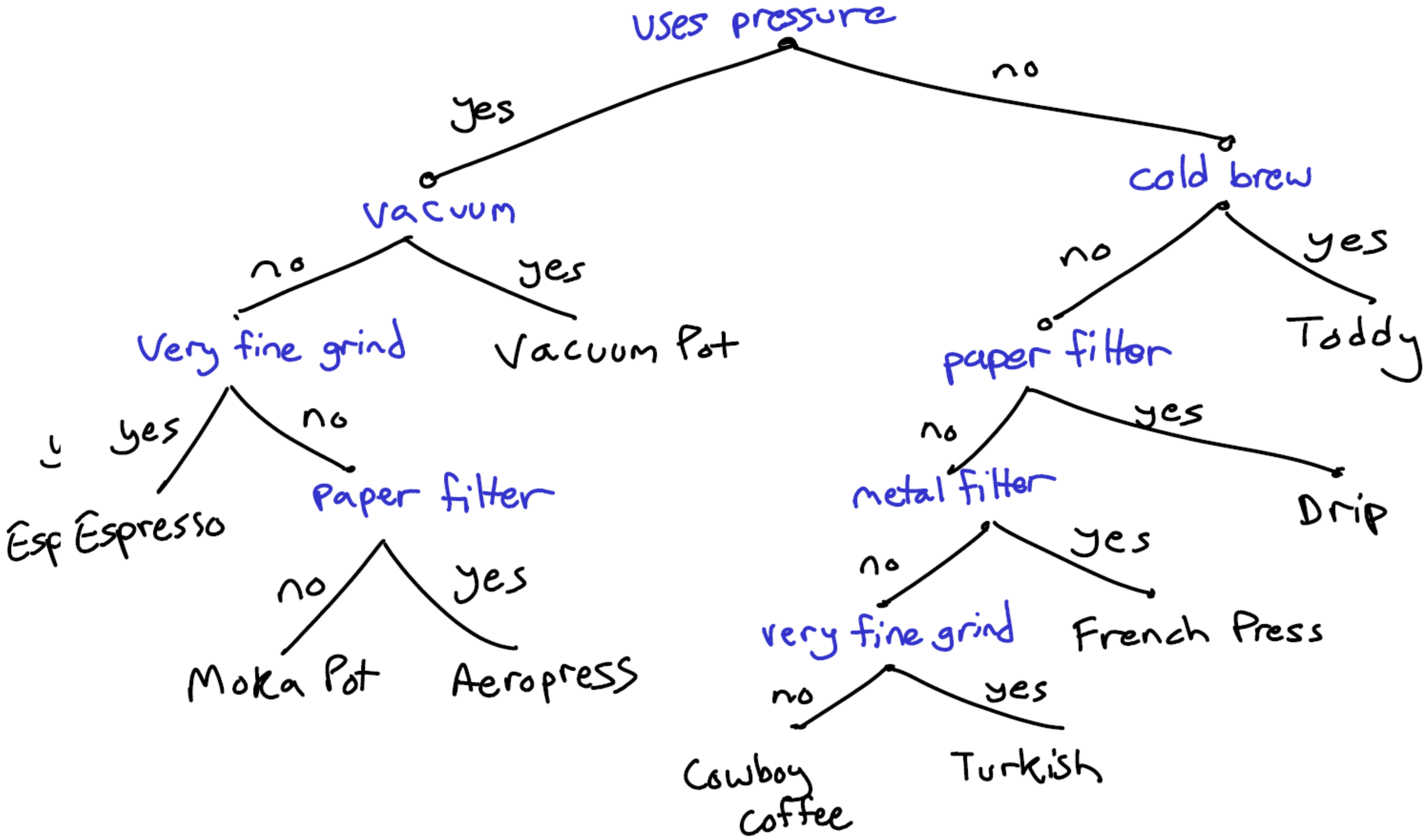
kNN Voting schemes

- Majority voting
  - Choose majority class amongst k closest neighbors
- Weighted voting
  - Weight each of the k neighbors' labels according to the distance to the training instance
  - In principle, this can be applied to an all-neighbors approach

Decision Tree Classifier

- Build a tree where each node represents a test
  - Decision tree: leaf nodes assign labels
  - Regression tree: leaf nodes assign real values
- Decide quality measure for choosing branching features
- Building the tree is expensive, but testing is fast
- Overfitting the data can be a problem

Coffee-makers



Building the tree

- Choose feature that is most discriminative across the training set
  - Information gain is commonly used
- Split the training data according to this feature
- Repeat for each subset of data
- Stop at some threshold

Information Theory

- A stochastic look at “information”
- The more uncertain a system, the more bits are required to describe it

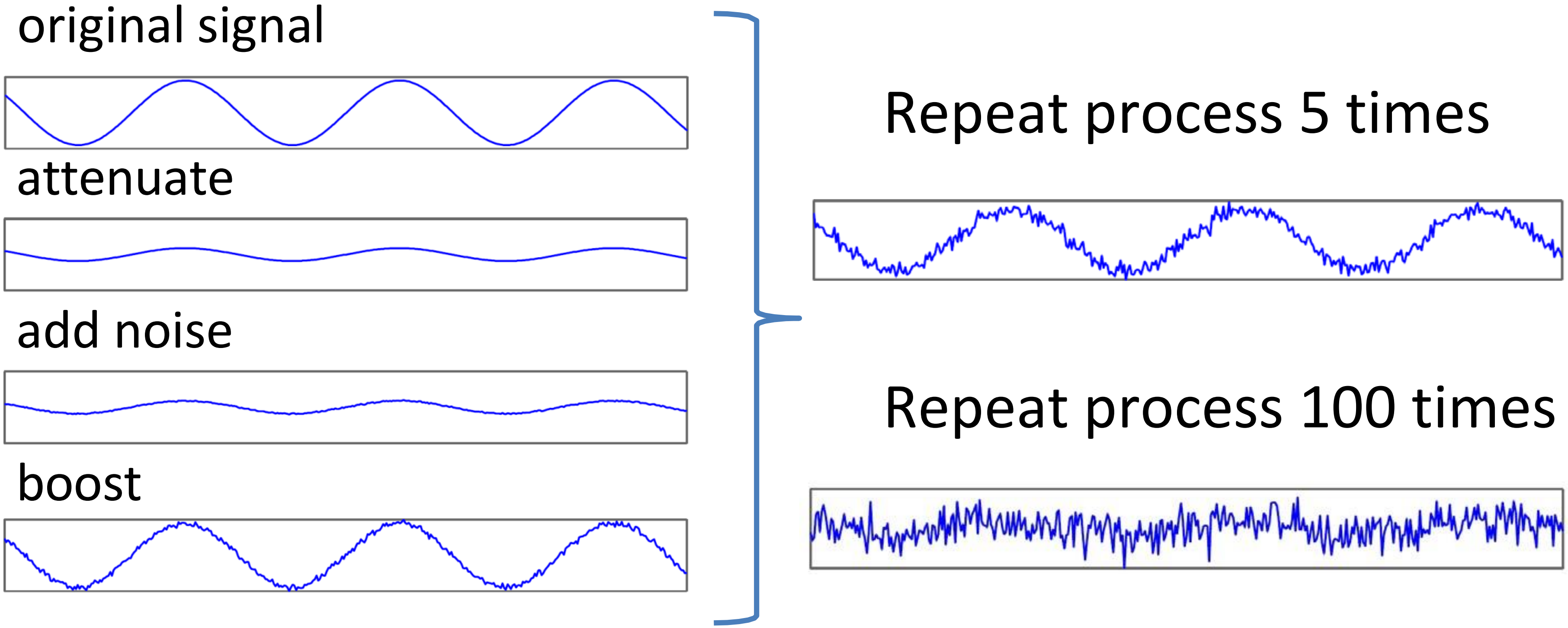
following slides from Rob Malouf

Information Theory

Claude Shannon. 1948. *A mathematical theory of communication*.

"The fundamental problem of communication is that of **reproducing** at one point… a message **selected** at another point… Semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one **selected from a set** of possible messages."

Information theory



original signal

attenuate

add noise

boost

Repeat process 5 times

Repeat process 100 times

Information theory

- Digital communications involves the transfer of symbols
- drawn from a discrete alphabet
  - English letters
  - English words
  - Decimal digits
  - Binary digits
  - DNA sequences
  - Quantized analog signals

Encoding information

- Minimal “piece” of information is one bit
- A bit can take on two values: { 0, 1 }
- There are 2^{bb} ways to arrange bb bits
- Therefore the number of bits required to encode ll different sequences is:

$$\lceil \log_2 ll \rceil$$

Example

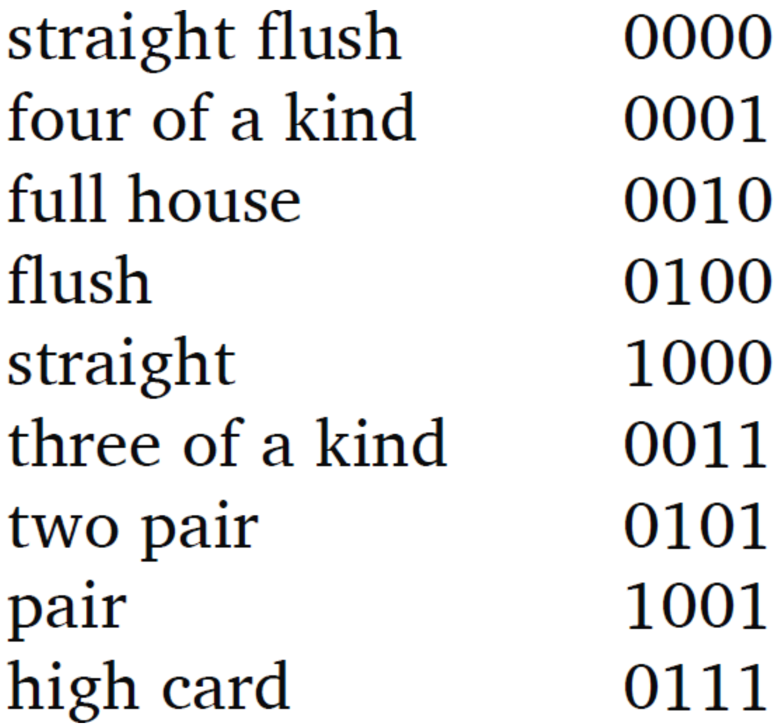
- Transmit information about a poker hand

{ straight flush, four of a kind, full house, flush, straight, three of a kind, two pair, pair, high card }

- There are 9 "messages"
- Baseline message length:

$$\lceil \log_2 9 \rceil = 4$$

Binary code for poker hands

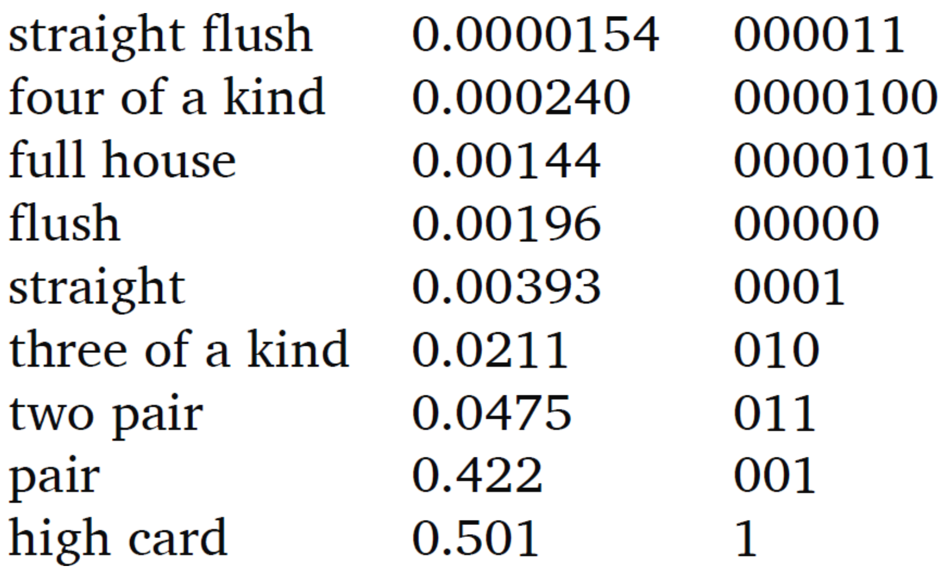


straight flush	0000
four of a kind	0001
full house	0010
flush	0100
straight	1000
three of a kind	0011
two pair	0101
pair	1001
high card	0111

Note: Some messages (e.g. 0110, 1010…) are unused; suggesting that there is waste in this encoding

Prefix encoding

- Probabilities can be used to reduce the expected message length

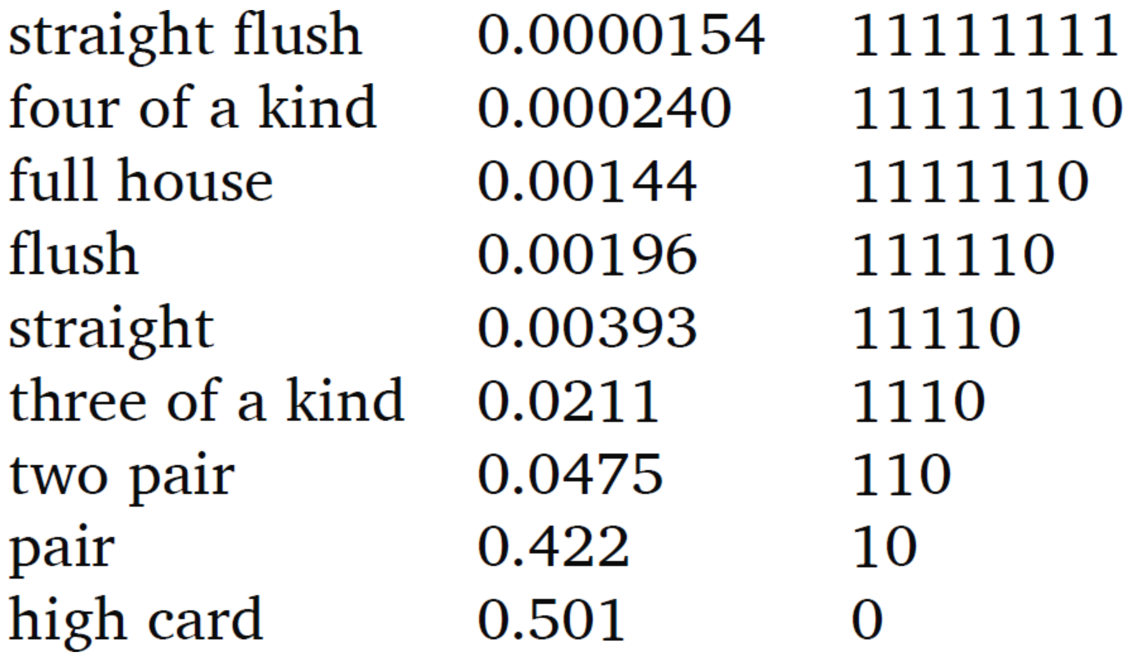


straight flush	0.0000154	000011
four of a kind	0.000240	0000100
full house	0.00144	0000101
flush	0.00196	00000
straight	0.00393	0001
three of a kind	0.0211	010
two pair	0.0475	011
pair	0.422	001
high card	0.501	1

- Now the expected length is reduced from 4 bits to 2.01 bits

Encoding information

- This encoding is even better



straight flush	0.0000154	11111111
four of a kind	0.000240	11111110
full house	0.00144	1111110
flush	0.00196	111110
straight	0.00393	11110
three of a kind	0.0211	1110
two pair	0.0475	110
pair	0.422	10
high card	0.501	0

- Here, the expected number of bits per hand is 1.61
- Can we do better? How would we find out?

Information and probability

- The information encoded is the identity of the poker hand
- The length of the message ought to be related to its information content
- A message that the opponent only has a pair or high card seems less informative than a message that they have four of a kind
  - because it happens more often
- Transmitting rare messages is more informative than transmitting common ones
- How can we make this more precise?

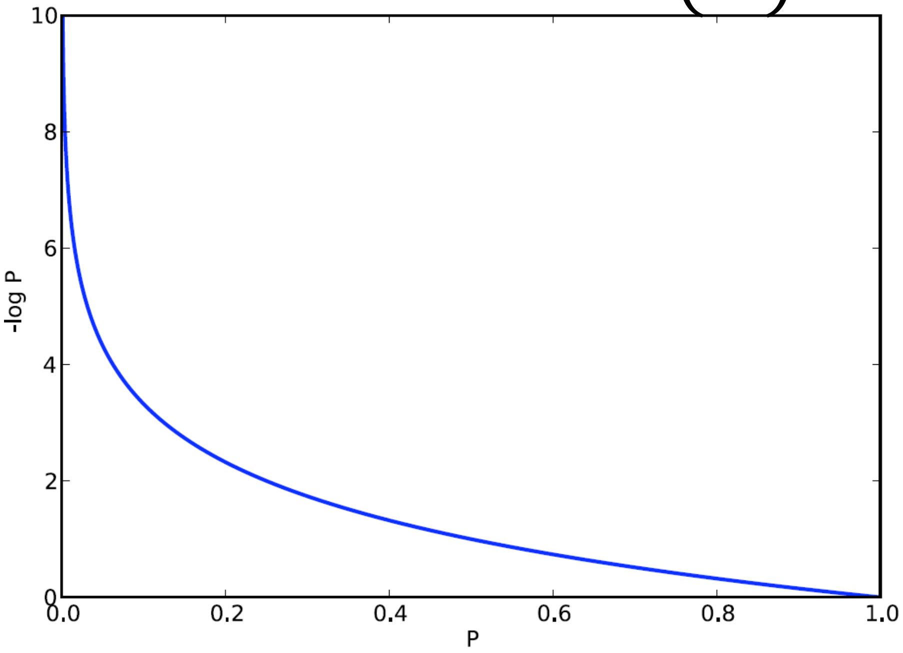
Information

- Let's assume information content of a message $II(pp)$ is a function of its probability
- Some basic properties that it seems like $II(pp)$ should have:
  - Information is non-negative $II(pp) \geq 0$
  - Certain messages contain no information $II(1) = 0$
  - Information should be additive for jointly occurring independent messages
  - $II(pp)$ should monotonically decrease versus pp

Information

- One math function which matches these criteria is

$$II(tt) - \log_{bb} pp(tt)$$



the base bb doesn't matter too much, because it just changes the measure by a constant factor

For $bb = 2$ we are measuring in bits
For $bb = tt$ we are measuring in nats
For $bb = 10$ we are measuring in hartleys

Entropy

- For the information content a message or a whole system, which is called its entropy, we sum over all possible messages or states
  - If we knew in advance which message we're selecting, we wouldn't need to code it

Entropy

The measure of uncertainty in a system

$$HH(XX) = - \text{?}\, PP(tt) \log PP(tt)$$

Information Theory

- Joint Entropy

$$HH(XX,YY) = -\sum_{xx}\sum_{yy} PP(tt,yy)\,(\log PP(tt,yy))$$

- Conditional Entropy

$$HH(YY|XX) = HH(XX,YY) - HH(XX)$$

- Mutual Information (or Information Gain): the expected reduction in entropy due to knowing something

$$\mathrm{IG}(YY|XX) = HH(YY) - HH(YY|XX)$$

Source Coding Theorem (Shannon)

The expected code length $EE[CC]$ for a random variable XX under an optimal encoding is

$$HH(XX) \leq EE[CC] \leq HH(XX) + 1$$

- This gives a lower bound for lossless compression and cryptography
- Guarantees that there is such an encoding
- Establishes the link between a probability distribution and information representation
- For the poker hands, $HH(XX) = 1.42$

Maximum Entropy Classification

- Model what is known; assume nothing about what is unknown
- Find a distribution that maximizes entropy (assumes the least)

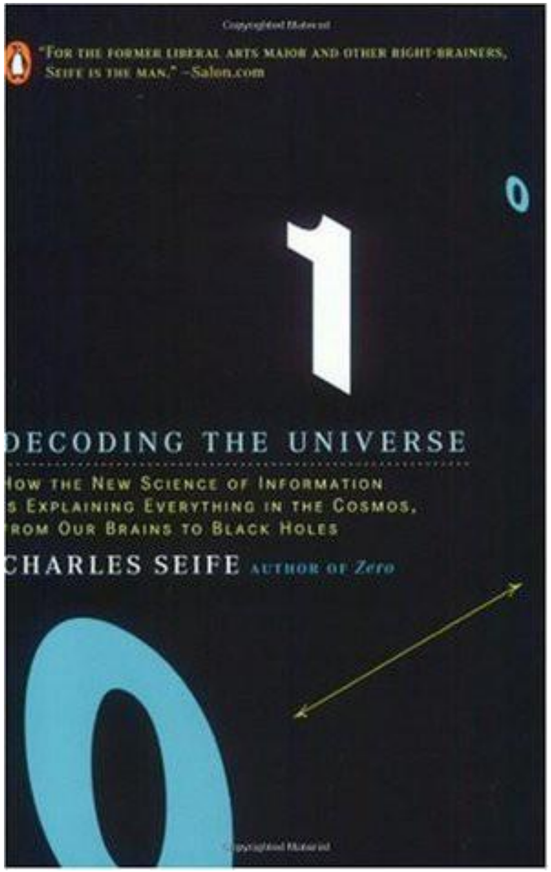
$$PP(yy|tt) = \frac{tt^{\sum_{jj} \lambda\lambda_{jj} ff_{jj}(xx,yy)}}{ZZ}$$

Training: Try to estimate $\lambda\lambda_{jj}$ such that

$$pp^* = \text{argmax}_{pp \in PP} HH(pp)$$

Testing: Evaluate $PP(yy|tt)$

Further information on… information theory


Copyrighted Material
"FOR THE FORMER LIBERAL ARTS MAJOR AND OTHER RIGHT-BRAINERS, SEIFE IS THE MAN." –Salon.com
0
1
DECODING THE UNIVERSE
HOW THE NEW SCIENCE OF INFORMATION IS EXPLAINING EVERYTHING IN THE COSMOS, FROM OUR BRAINS TO BLACK HOLES
CHARLES SEIFE AUTHOR OF Zero
0
Copyrighted Material

Decoding the Universe: How the New Science of Information Is Explaining Everything in the Cosmos, from Our Brains to Black Holes
January 30, 2007
by Charles Seife

note: this is a popular science recommendation, not a textbook or academic treatment

Dynamic Programming

Some material from:
Andrew McCallum, William Cohen

Overlapping sub-problems

- Often, a problem can be divided into sub-problems that interact with each other
- What's the longest substring that can be found between two strings?

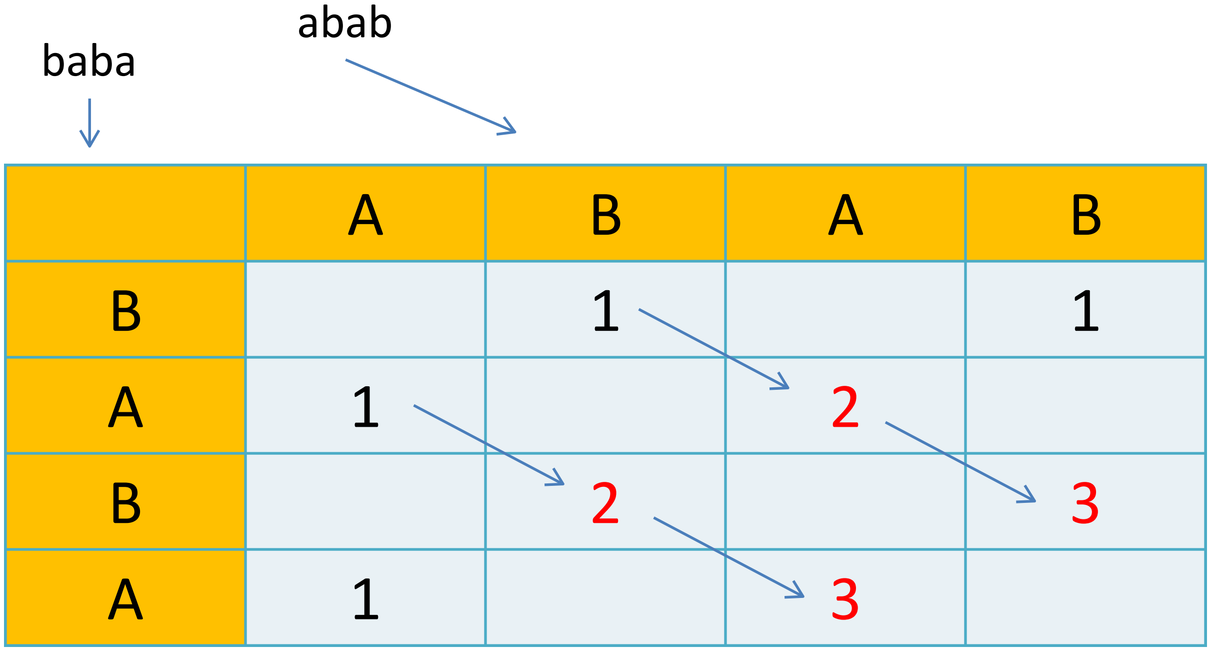
Find the longest common substring of *abab* and *baba*. (There are 2 of length 3)

```
abab    abab
 baba  baba
```

Longest substring

- When we can identify smaller subproblems, it makes sense to save these results and re-use them
- Let’s keep a table containing the lengths of the longest common *suffix* for every possible alignment of the two strings
- First, we’ll look at what doesn’t work…

Longest substring: non-dynamic



abab

baba

	A	B	A	B
B		1		1
A	1		2	
B		2		3
A	1		3	

We’re scanning the matrix too many times and changing values that we’ve already set. This is not going to work

Longest substring: dynamic

	A	B	A	B
B	0	1	0	1
A	1	0	2	0
B	0	2	0	3
A	1	0	3	0

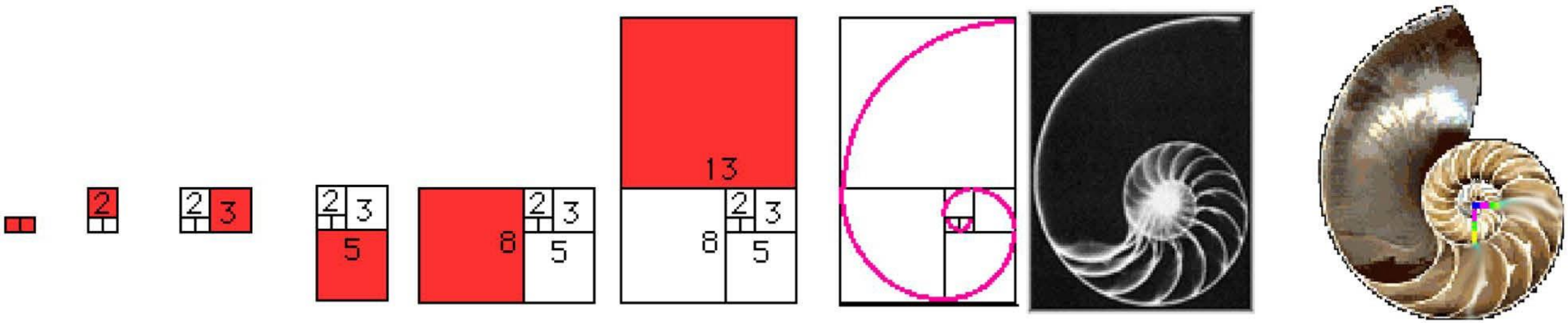
[redacted]

By calculating the cells in a certain order, we are able to incorporate previous results into each calculation

Fibonacci numbers

$$F_n = F_{n-2} + F_{n-1}$$

$$F_1 = F_2 = 1$$



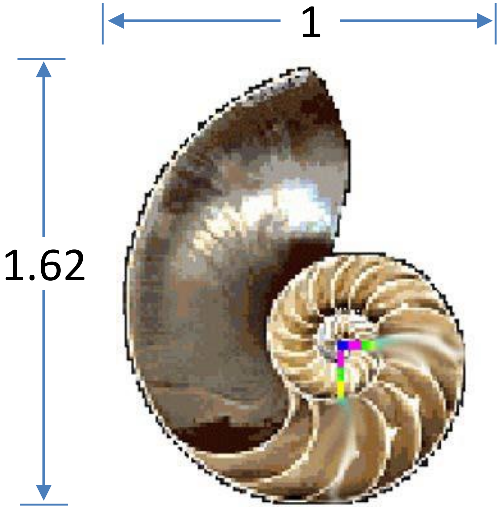
1 1 2 3 5 8 13 21 34 …

Golden ratio

$$F_n = F_{n-2} + F_{n-1}$$

$$F_1 = F_2 = 1$$

$$ll = \frac{F_{n+1}}{F_n} \approx 1.62$$



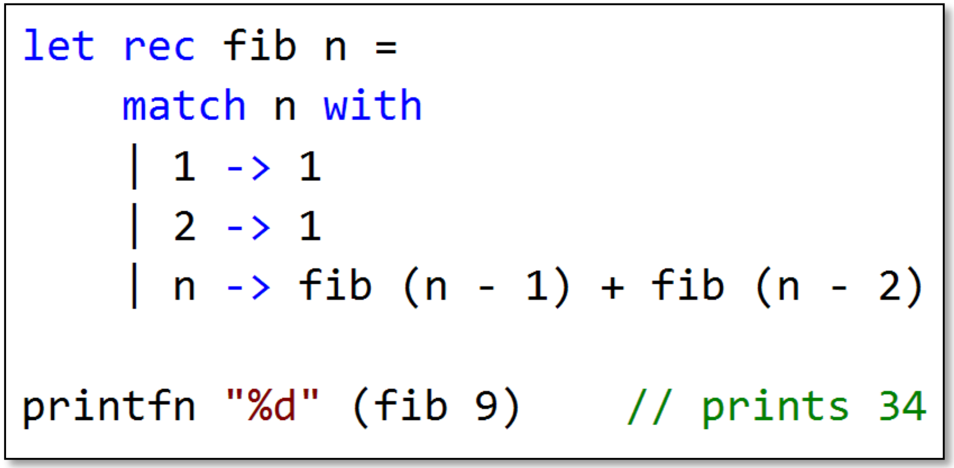
Programming the Fibonacci sequence

- Mathematicians are happy to write:

$$F_n = F_{n-2} + F_{n-1}$$

$$F_1 = F_2 = 1$$

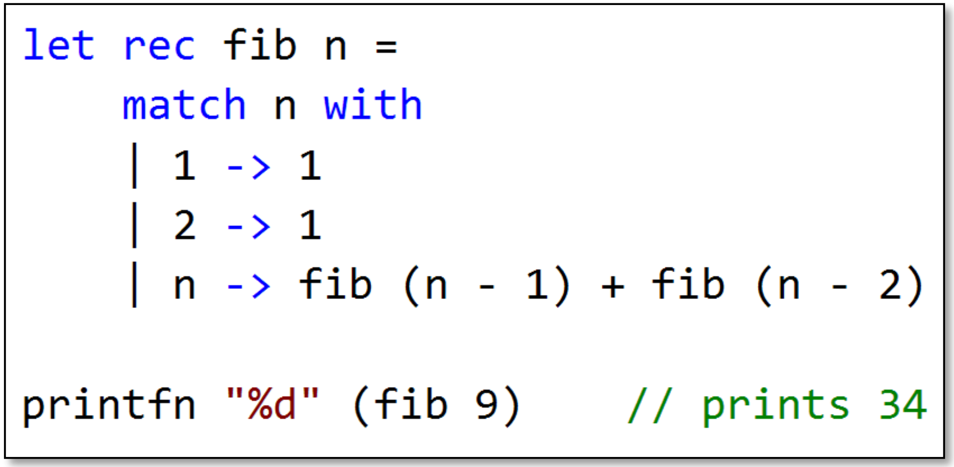
...and call it a day. This why they like functional programming languages (i.e. F#):



```
let rec fib n =
    match n with
    | 1 -> 1
    | 2 -> 1
    | n -> fib (n - 1) + fib (n - 2)

printfn "%d" (fib 9)     // prints 34
```

Fibonacci: non-dynamic

An F# function to return the ll-th Fibonacci number:

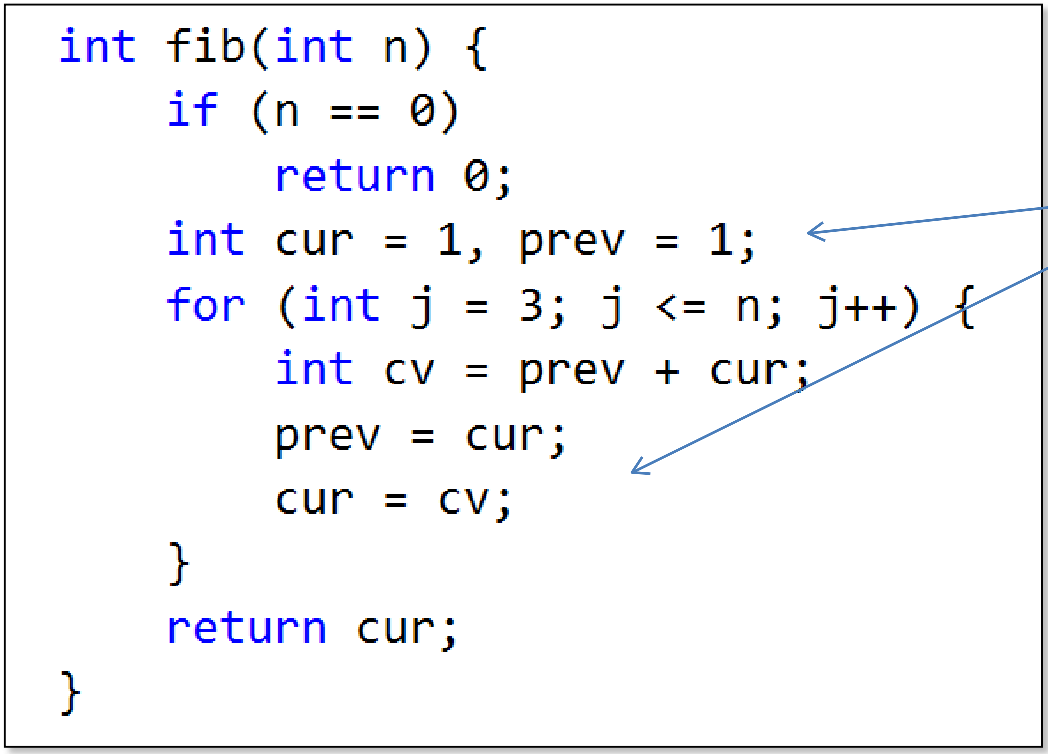


```
let rec fib n =
    match n with
    | 1 -> 1
    | 2 -> 1
    | n -> fib (n - 1) + fib (n - 2)

printfn "%d" (fib 9)     // prints 34
```

A naïve, implementation of this would have time complexity $OO(2ll!)$. Fortunately, F# is smarter than that behind-the-scenes.

Fibonacci: dynamic programming!

A C, C++, C# function to return the ll-th Fibonacci number:



```
int fib(int n) {
    if (n == 0)
        return 0;
    int cur = 1, prev = 1;
    for (int j = 3; j <= n; j++) {
        int cv = prev + cur;
        prev = cur;
        cur = cv;
    }
    return cur;
}
```

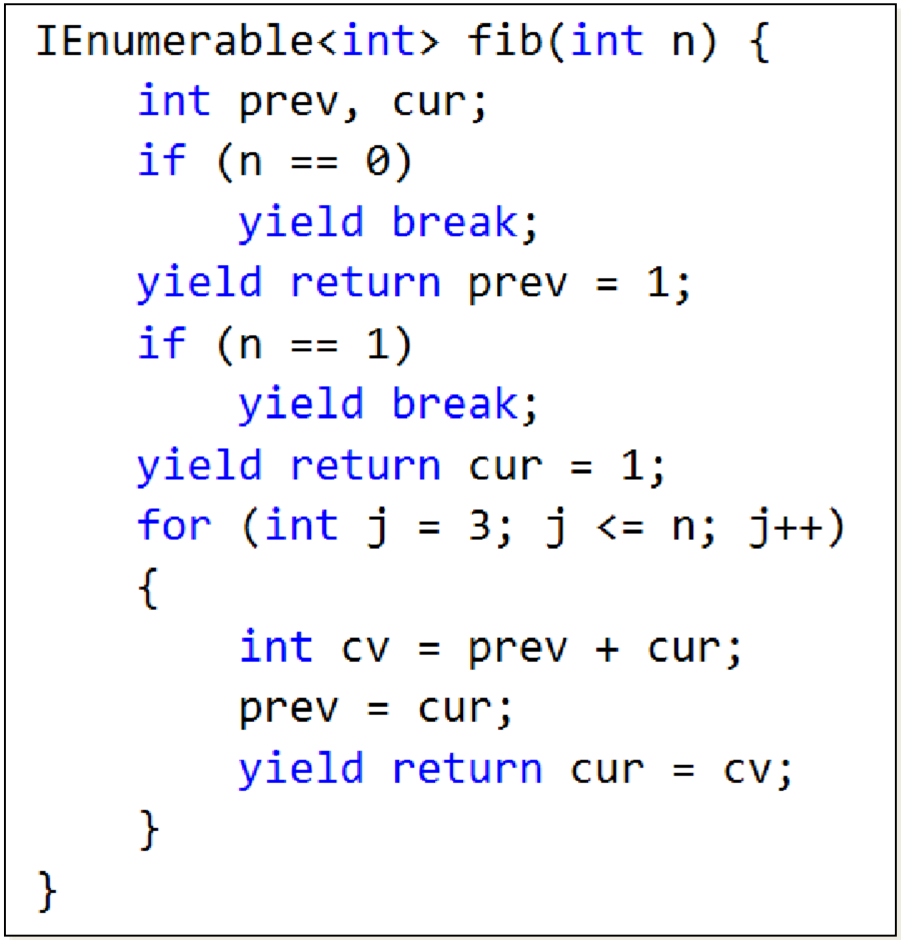
Saving two values as we go along, so we don't need to call recursively (twice)

This is called dynamic programmin
g

Fibonacci dynamic programming

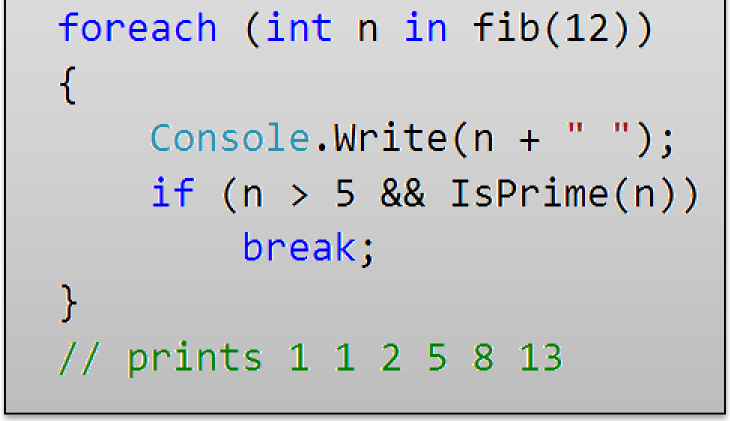
- On the previous slide, we only needed to save two values. What if we want the whole sequence of Fibonacci numbers from 1 … F_n?
- Well, the best way would be to provide them on demand
  - This won't illustrate the dynamic programming but please allow the digression
  - We'll modify the previous function to yield values as it goes along.
  - We use a C# iterator, which illustrates deferred execution (it calculates values one-at-a-time, only as needed)

Deferred execution iterator



```
IEnumerable<int> fib(int n) {
    int prev, cur;
    if (n == 0)
        yield break;
    yield return prev = 1;
    if (n == 1)
        yield break;
    yield return cur = 1;
    for (int j = 3; j <= n; j++)
    {
        int cv = prev + cur;
        prev = cur;
        yield return cur = cv;
    }
}
```

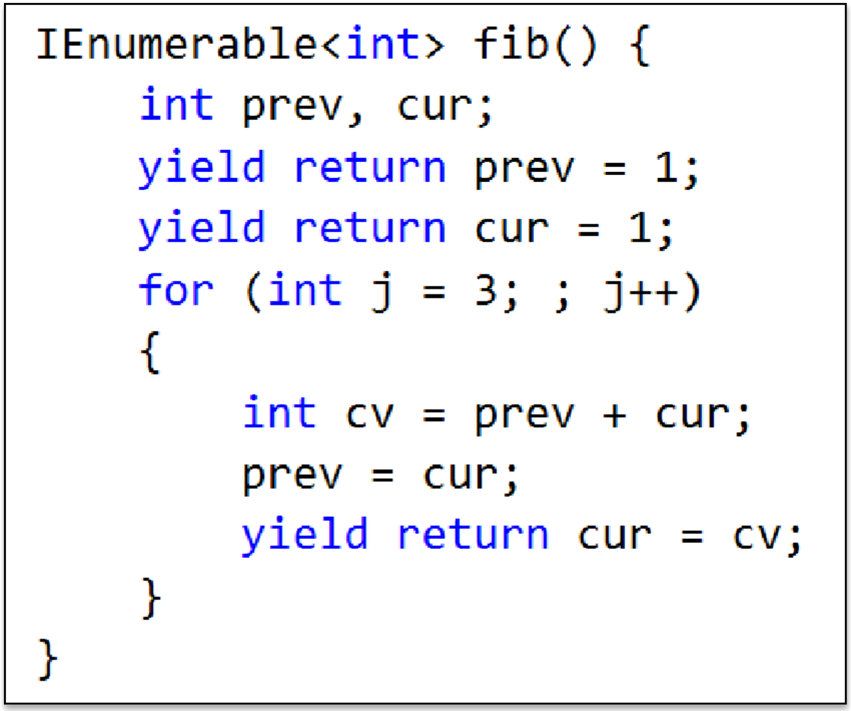
This is nice; if the caller changes his mind halfway through, and doesn't need all *ll* numbers, there will be no wasted work



```
foreach (int n in fib(12))
{
    Console.Write(n + " ");
    if (n > 5 && IsPrime(n))
        break;
}
// prints 1 1 2 5 8 13
```

This suggests that our deferred execution fib function *shouldn't even care* about *ll*…

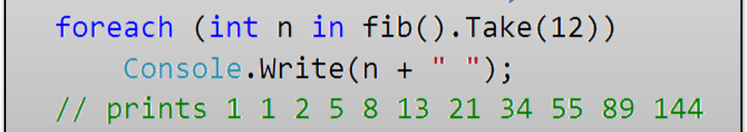
Deferred execution Fibonacci



```
IEnumerable<int> fib() {
    int prev, cur;
    yield return prev = 1;
    yield return cur = 1;
    for (int j = 3; ; j++)
    {
        int cv = prev + cur;
        prev = cur;
        yield return cur = cv;
    }
}
```

Recall: an iterator is a special function containing the yield keyword.
It must return an IEnumerable<T>

Now the caller can decide how many Fibonacci numbers she wants. In fact, she doesn't even need to know or decide in advance.

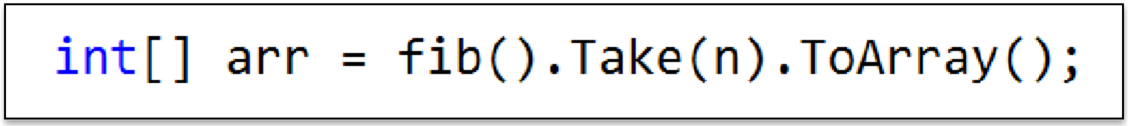


```
foreach (int n in fib().Take(12))
    Console.Write(n + " ");
// prints 1 1 2 5 8 13 21 34 55 89 144
```

Take is a system-defined Linq operator that returns only the first
ll elements of any sequence (or fewer if the sequence ends before

Now, back to dynamic programming?

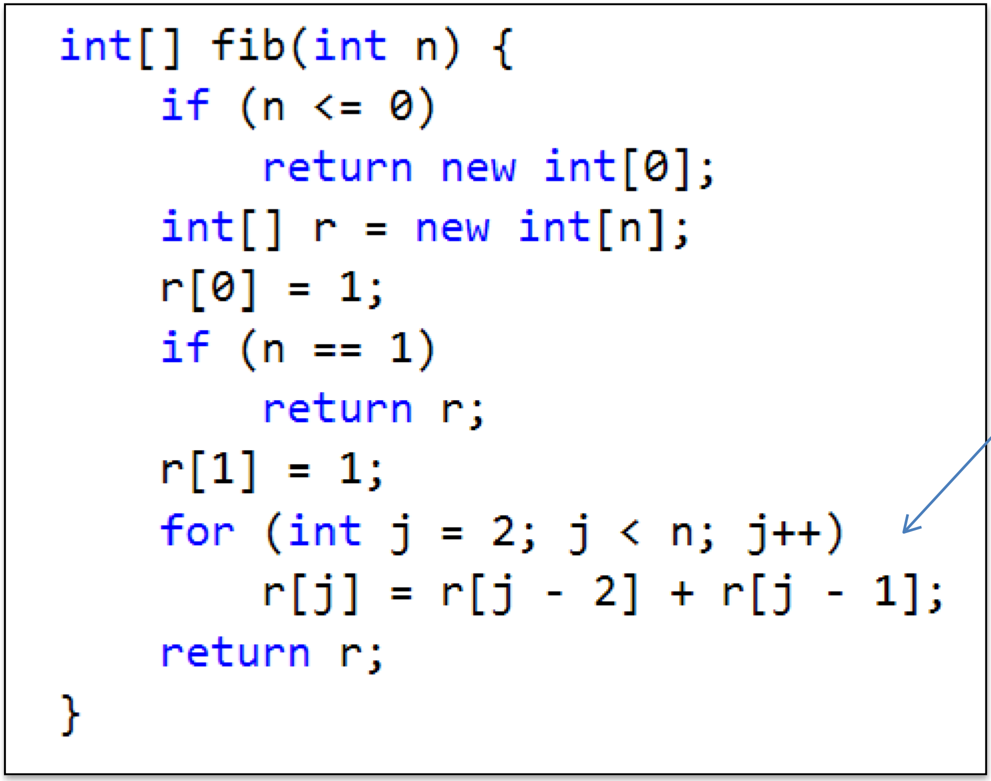
- Suppose we require an *actual array* of the first ll Fibonacci numbers. That is, we'll be needing random access to them.
  - Once again, we're distracted by Linq:



```
int[] arr = fib().Take(n).ToArray();
```

  - That's simple but it doesn't illustrate dynamic programming
  - So let's show the dynamic programming array version

Fibonacci numbers: dynamic programming



```
int[] fib(int n) {
    if (n <= 0)
        return new int[0];
    int[] r = new int[n];
    r[0] = 1;
    if (n == 1)
        return r;
    r[1] = 1;
    for (int j = 2; j < n; j++)
        r[j] = r[j - 2] + r[j - 1];
    return r;
}
```

Once we get started, the calculation is simple because we keep all previous results

This is the idea behind dynamic programming. Save the results of calculations that you (might) need later.

Dynamic programming

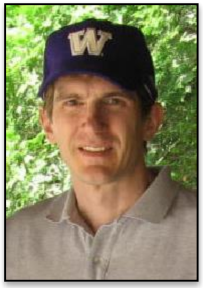
- We see this pattern often in computational linguistics (and in Project 6)
  1. Create a table to hold solutions to the sub-problems
  2. Fill in the table, re-using these previous results

Edit distance

Given two sequences, return the distance as measured by: the *minimum* number of editing operations needed to turn the first sequence into the second.

Example: sequence of *characters* (a “string”)

```
Greg
Glenn
```



1. Substitute `r` to `l`
2. Substitute `g` to `n`
3. Add an `n`

Distance: 3

Edit distance

- Widely used in computational linguistics:
  - Spell checking
  - Error correction
  - Text alignment (diff)
  - Aligning parallel corpora for machine translation training
  - Word error rate in speech recognition

Levenshtein edit distance

- Given strings ss and tt:
  - The Levenshtein edit distance is the least-cost sequence of edit commands that transform ss to tt.
  - Costs from the original paper (Levenshtein 1966):

Copy (same character)	0
Delete	1
Insert	1
Substitute (different character)	1

  - With these costs; the function is commutative; it will has the same value for the reverse direction, tt to ss.

Substitution cost

- Levenshtein also proposed a version without substitution

	Copy (same character)	0
	Delete	1
	Insert	1
✗	~~Substitute (different character)~~	~~1~~

- So, to substitute, you insert (+1) and delete (+1) = 2

	Copy (same character)	0
	Delete	1
	Insert	1
✓	Substitute (different character)	**2**

Use this substitution cost for *string* edit distance in Project 6

Alignment

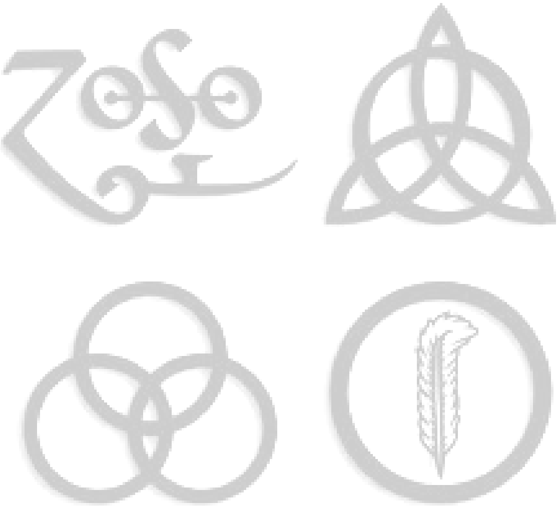
Some cases are easy:

```
Lead Zeppelin
Led Zeppelin
```

```
Lead Zeppelin

Le←d Zeppelin
```

1 delete operation

Notice that calculating the edit distance implies an alignment between the two strings:



```
Lead Zeppelin
|||||||||||||
Leλλd Zeppelin
```

Not so easy

```
Edith couldn’t stand her sister.
Who was standing by her mother?
```

- It’s not immediately clear what the edit distance is.
- It’s not clear how these should be aligned.
- What if we consider all possible alignments by brute force? How many are there?

Answer: A LOT!

How many alignments?

- This is a hard question to answer because it depends on whether you want different alignments to $\lambda\lambda$ to be considered distinct. An upper bound is given by:

$$\overset{\min m,n}{\underset{k\Box\ 0}{}} \quad \frac{m\Box\ n\Box k\Box}{k\Box\ m\Box\ k\Box\ n\Box\ k\Box}$$

- But a more relevant answer is probably

$$ll(mm,ll) = 2\left(ll(mm-1,ll-1) + \underset{ii=0}{\overset{nn-2}{?}} ll(mm-1,ii) + \underset{ii=0}{\overset{mm-2}{?}} ll(ii,ll-1)\right)$$

- Either of these is unusably huge
- For details, see www.ai.uga.edu/mc/number.pdf

example

```
ab
cd

ab
 cd

ab
cd

ab
  cd
```

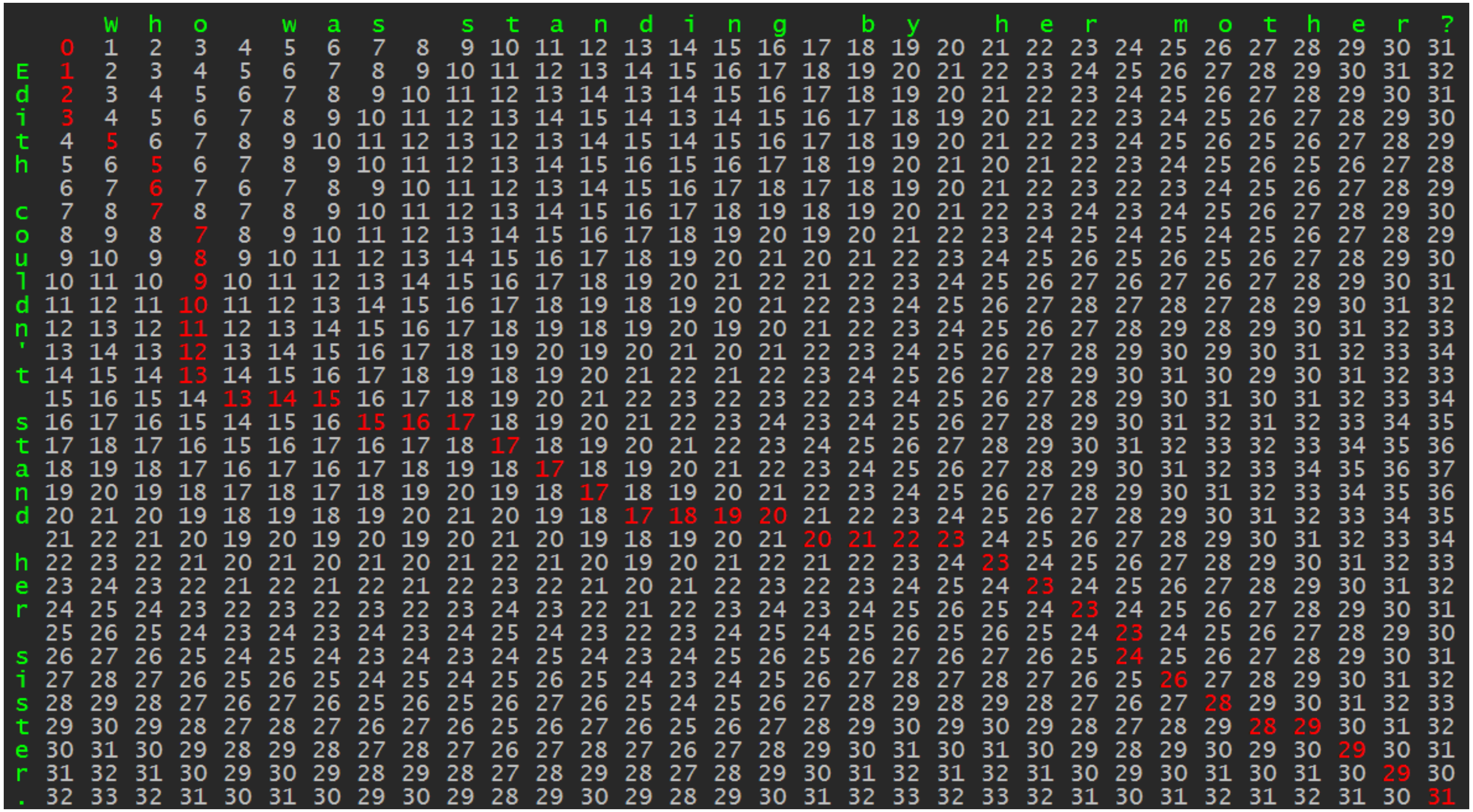
```
ab
c d

a b
cd

a b
 cd

a b
c d
```

```
a b
 c d

a  b
 cd

ab
c  d

etc...
```

Enumerating all possible alignments is definitely not going to work

There are too many!

Dynamic programming finds a solution in $00(ll^2)$



```
      W  h  o     w  a  s     s  t  a  n  d  i  n  g     b  y     h  e  r     m  o  t  h  e  r  ?
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
E   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
d   2  3  4  5  6  7  8  9 10 11 12 13 14 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
i   3  4  5  6  7  8  9 10 11 12 13 14 15 14 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
t   4  5  6  7  8  9 10 11 12 13 12 13 14 15 14 15 16 17 18 19 20 21 22 23 24 25 26 25 26 27 28 29
h   5  6  5  6  7  8  9 10 11 12 13 14 15 16 15 16 17 18 19 20 21 20 21 22 23 24 25 26 25 26 27 28
    6  7  6  7  6  7  8  9 10 11 12 13 14 15 16 17 18 17 18 19 20 21 22 23 22 23 24 25 26 27 28 29
c   7  8  7  8  7  8  9 10 11 12 13 14 15 16 17 18 19 18 19 20 21 22 23 24 23 24 25 26 27 28 29 30
o   8  9  8  7  8  9 10 11 12 13 14 15 16 17 18 19 20 19 20 21 22 23 24 25 24 25 24 25 26 27 28 29
u   9 10  9  8  9 10 11 12 13 14 15 16 17 18 19 20 21 20 21 22 23 24 25 26 25 26 25 26 27 28 29 30
l  10 11 10  9 10 11 12 13 14 15 16 17 18 19 20 21 22 21 22 23 24 25 26 27 26 27 26 27 28 29 30 31
d  11 12 11 10 11 12 13 14 15 16 17 18 19 18 19 20 21 22 23 24 25 26 27 28 27 28 27 28 29 30 31 32
n  12 13 12 11 12 13 14 15 16 17 18 19 18 19 20 19 20 21 22 23 24 25 26 27 28 29 28 29 30 31 32 33
'  13 14 13 12 13 14 15 16 17 18 19 20 19 20 21 20 21 22 23 24 25 26 27 28 29 30 29 30 31 32 33 34
t  14 15 14 13 14 15 16 17 18 19 18 19 20 21 22 21 22 23 24 25 26 27 28 29 30 31 30 29 30 31 32 33
   15 16 15 14 13 14 15 16 17 18 19 20 21 22 23 22 23 22 23 24 25 26 27 28 29 30 31 30 31 32 33 34
s  16 17 16 15 14 15 16 15 16 17 18 19 20 21 22 23 24 23 24 25 26 27 28 29 30 31 32 31 32 33 34 35
t  17 18 17 16 15 16 17 16 17 18 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 32 33 34 35 36
a  18 19 18 17 16 17 16 17 18 19 18 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
n  19 20 19 18 17 18 17 18 19 20 19 18 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
d  20 21 20 19 18 19 18 19 20 21 20 19 18 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
   21 22 21 20 19 20 19 20 19 20 21 20 19 18 19 20 21 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
h  22 23 22 21 20 21 20 21 20 21 22 21 20 19 20 21 22 21 22 23 24 23 24 25 26 27 28 29 30 31 32 33
e  23 24 23 22 21 22 21 22 21 22 23 22 21 20 21 22 23 22 23 24 25 24 23 24 25 26 27 28 29 30 31 32
r  24 25 24 23 22 23 22 23 22 23 24 23 22 21 22 23 24 23 24 25 26 25 24 23 24 25 26 27 28 29 30 31
   25 26 25 24 23 24 23 24 23 24 25 24 23 22 23 24 25 24 25 26 25 26 25 24 23 24 25 26 27 28 29 30
s  26 27 26 25 24 25 24 23 24 23 24 25 24 23 24 25 26 25 26 27 26 27 26 25 24 25 26 27 28 29 30 31
i  27 28 27 26 25 26 25 24 25 24 25 26 25 24 23 24 25 26 27 28 27 28 27 26 25 26 27 28 29 30 31 32
s  28 29 28 27 26 27 26 25 26 25 26 27 26 25 24 25 26 27 28 29 28 29 28 27 26 27 28 29 30 31 32 33
t  29 30 29 28 27 28 27 26 27 26 25 26 27 26 25 26 27 28 29 30 29 30 29 28 27 28 29 28 29 30 31 32
e  30 31 30 29 28 29 28 27 28 27 26 27 28 27 26 27 28 29 30 31 30 31 30 29 28 29 30 29 30 29 30 31
r  31 32 31 30 29 30 29 28 29 28 27 28 29 28 27 28 29 30 31 32 31 32 31 30 29 30 31 30 31 30 29 30
.  32 33 32 31 30 31 30 29 30 29 28 29 30 29 28 29 30 31 32 33 32 33 32 31 30 31 32 31 32 31 30 31
```

Edith couldn’t λλλλsλλλλtandλλλλλλ λλλλλλher sistλλer.
λλλλλλWhλλλλoλλλλλλλλλλλλ was standing by her λλmother?

Dynamic programming for edit distance

- Given strings $ss_{1\ldots jj\ldots mm}$ and $tt_{1\ldots ii\ldots nn}$
- Define an $(ll+1)$-column by $(mm+1)$-row rectangular array where each cell $[ii, jj]$ contains the number of edit operations needed to align $ss_{1\ldots jj}$ with $tt_{1\ldots ii}$.

Jurafsky and Martin notation from Figure 3.25, p. 76

	Source sequence ss	**Target sequence tt**
sequence length:	mm	ll
sequence index:	jj $ss[jj]$	ii $tt[ii]$
in the table:	each row → is an element dist[..., jj]	each column ↓ is an element dist[ii, ...]

Dynamic programming table for string edit distance

measure the edit distance between STORE and SOUR

target

source

	λλ	**S**	**O**	**U**	**R**
λλ	0	1	2	3	4
S	1				
T	2				
O	3				
R	4				
E	5				

objective: fill each cell with the number of edit operations needed to align $ss_{1\ldots jj}$ with $tt_{1\ldots ii}$

Subdivide the problem

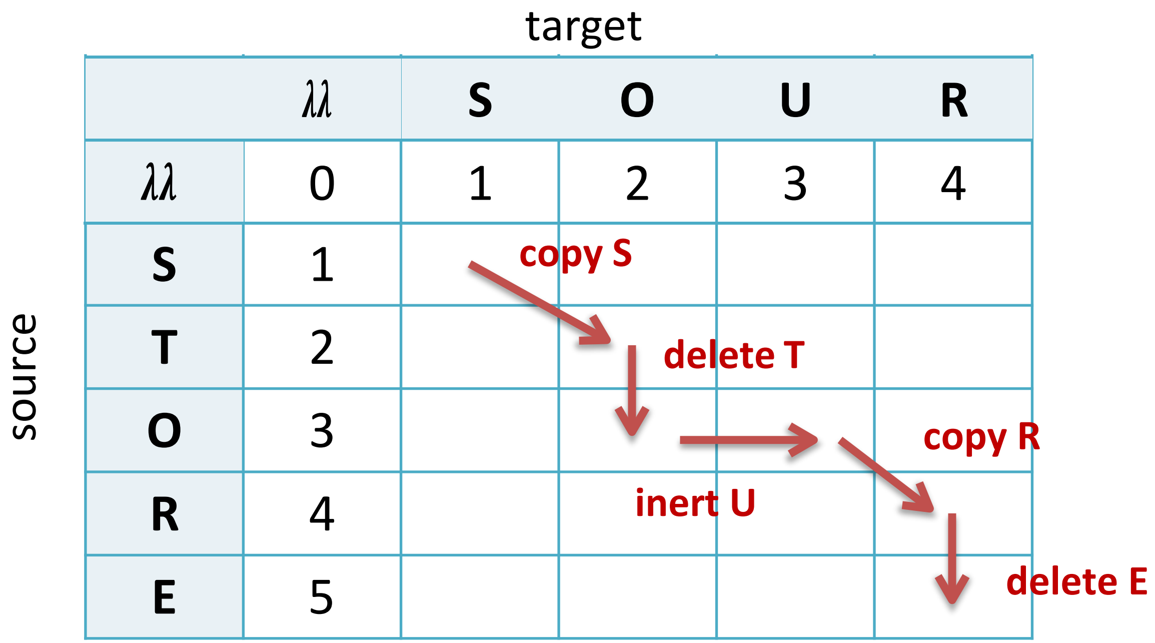
- Given a partial solution, the next incremental step is easy
- Partial solution:

  We have: the cost for aligning $ss_{1\ldots jj}$ with $tt_{1\ldots ii}$

- Next step:

  To align $ss_{1\ldots jj+1}$ with $tt_{1\ldots ii}$, would the last operation be a copy (0), substitute (2), insert (1), or delete (1)?

How the table works



target

source

	λλ	S	O	U	R
λλ	0	1	2	3	4
S	1				
T	2				
O	3				
R	4				
E	5				

Alignment:

STOλλRE
||||||
SλλOURλλ

When going from source to target:

→ A horizontal arrow is an insertion

↓ A vertical arrow is a deletion

↘ A diagonal arrow is a copy or substitution

example

start by putting a zero here

target

source

	λλ	S	O	U	R
λλ	0	1	2	3	4
S	1				
T	2				
O	3				
R	4				
E	5				

Notice the table has an extra row and an extra column. These are initialized with the edit distance between the empty string *λλ* and the first *jj* or *ii* characters of the source or target string, respectively

example

Now we want to enter the best (minimum) cost for the first cell. There are 3 operations to consider.

target

source

	λλ	S	O	U	R
λλ	0	1	2	3	4
S	1	?			
T	2				
O	3				
R	4				
E	5				

1. Insert (cost: 1)
2. Delete (cost: 1)
3. Copy or substitute (cost: 0 or 2)

The new cost is added to the cost (thus far) for the cell you are coming from

example

Examine the source and target characters to determine if option (3.) is a source or copy, and use the corresponding cost.

target

source

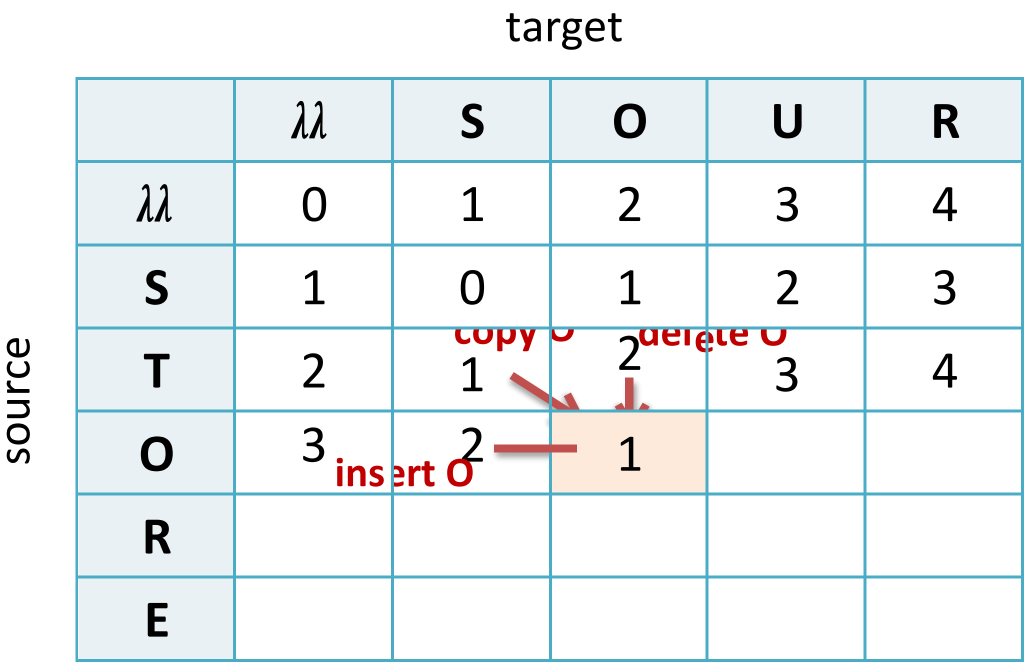
	$\lambda\lambda$	**S**	**O**	**U**	**R**
$\lambda\lambda$	0	1	2	3	4
S	1	0			
T	2				
O	3				
R	4				
E	5				

In this case option (3) is a **copy** (of the character ‘S’), so the incremental cost is zero.

$$\min(iillss, ddttll, ccccppyy) = \min(1+1, 1+1, 0+0) = 0$$

example

Continue from left-to-right (columns) and top-to-bottom (rows) filling in values


target

source

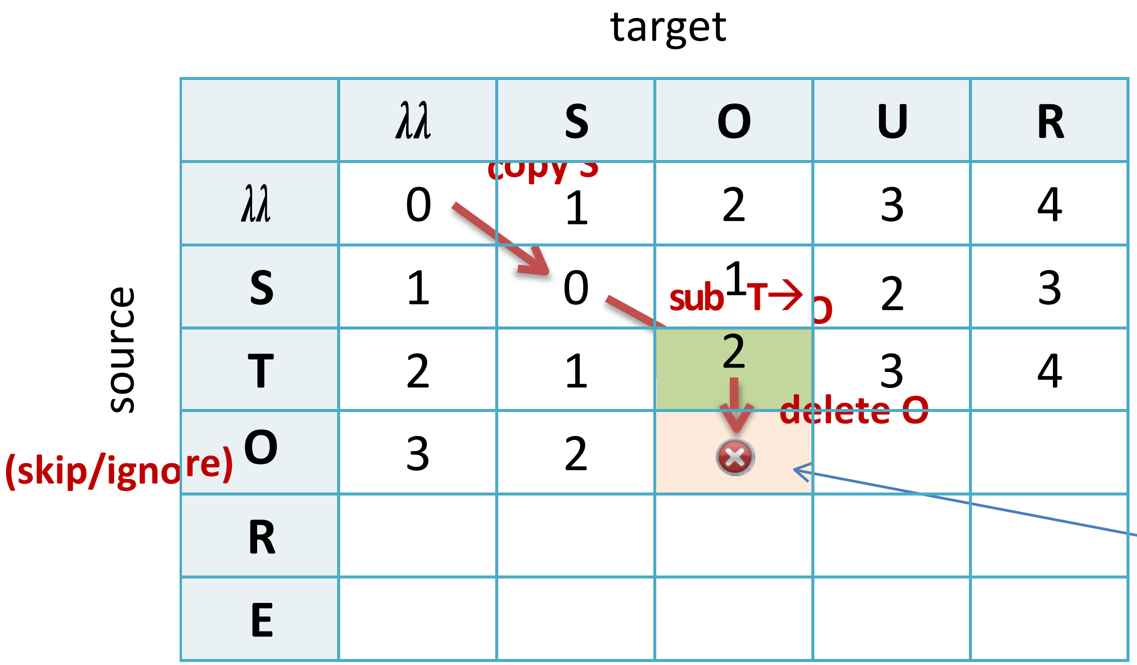
	λλ	S	O	U	R
λλ	0	1	2	3	4
S	1	0	1	2	3
T	2	1	2	3	4
O	3	2	1		
R					
E					

Compare:

target	operation	cost
Sλλ	copy O	0
Sλλ	insert O	1
SO	delete O	1

here, copying ‘o’ has the lowest cost

Consider the ‘delete O’ case. Why isn’t it ‘delete T’?
If we got to the green cell, we probably substituted T→O for the second step. Now we’re considering throwing *that* O away and copying O from the target (which skips the *source* O).



	λλ	S	O	U	R
λλ	0	1	2	3	4
S	1	0	1	2	3
T	2	1	2	3	4
O	3	2	☒		
R					
E					

target	operation	cost
Sλλ	copy O	0
Sλλ	insert O	1
SO	delete O	1

Naturally, we don’t choose this for the new cell, since it has a cost of 3. This whole line of inquiry ends

example

When you’re done, the minimum edit distance is the value in cell $[ll, mm]$.

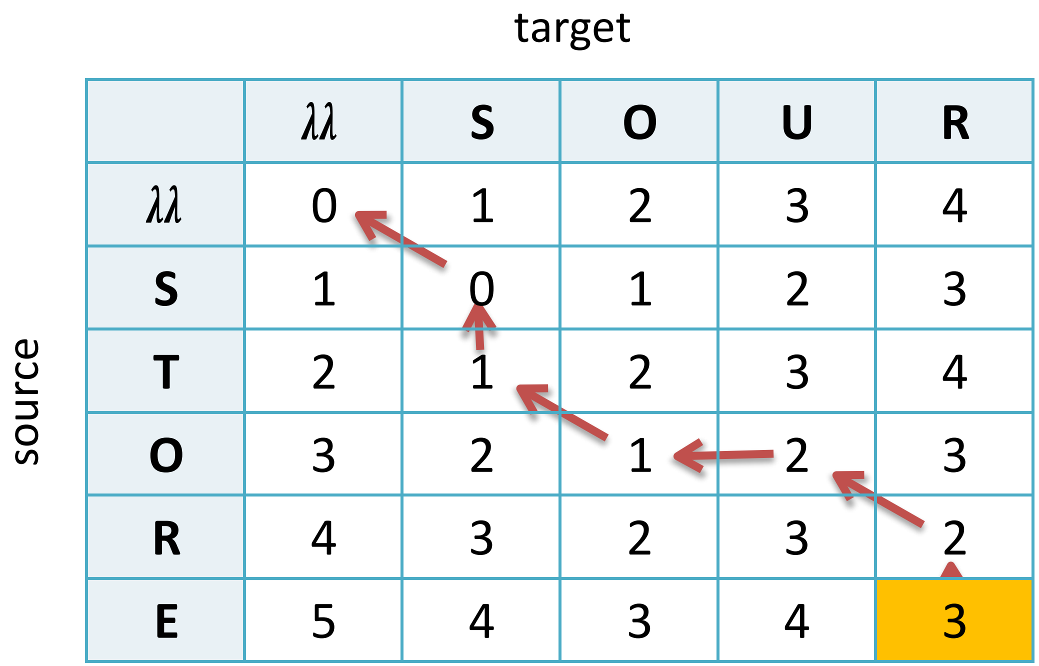
target

	λλ	**S**	**O**	**U**	**R**
λλ	0	1	2	3	4
S	1	0	1	2	3
T	2	1	2	3	4
O	3	2	1	2	3
R	4	3	2	3	2
E	5	4	3	4	3

The edit distance between STORE and SOUR is 3.

example

Starting from cell $[ll, mm]$, recover a backtrace. In a greedy fashion, take the path from $[ll, mm]$ to $[0, 0]$ by selecting the lowest-valued neighbor cell. In this case, there is only one.



target

source

	$\lambda\lambda$	S	O	U	R
$\lambda\lambda$	0	1	2	3	4
S	1	0	1	2	3
T	2	1	2	3	4
O	3	2	1	2	3
R	4	3	2	3	2
E	5	4	3	4	3

Edit distance

$$DD_{ii,jj} = \min \begin{cases} DD_{ii-1,jj-1} + 0 & (ss_{jj} = tt_{ii}) \quad ccccppyy \\ DD_{ii-1,jj-1} + 2 & (ss_{jj} \neq tt_{ii}) \quad ssssbbsstt. \\ DD_{ii-1,jj} + 1 & \quad iillssttiitt \\ DD_{ii,jj-1} + 1 & \quad ddttlltttttt \end{cases}$$

$$DD_{ii,0} = ii$$

$$DD_{0,jj} = jj$$

$$dd = DD_{nn,mm}$$

Edit distance

$$DD_{ii,jj} = \min \begin{cases} DD_{ii-1,jj-1} + 0 & (ss_{jj} = tt_{ii}) \quad \mathit{ccccppyy} \\ DD_{ii-1,jj-1} + 2 & (ss_{jj} \neq tt_{ii}) \quad \mathit{sssbbsstt}. \\ DD_{ii-1,jj} + 1 & \quad \mathit{iillssttiitt} \\ DD_{ii,jj-1} + 1 & \quad \mathit{ddttllttttt} \end{cases}$$

$$DD_{ii,0} = ii$$

$$DD_{0,jj} = jj$$

$$dd = DD_{nn,mm}$$

$$dd_{nnnnnnnmm} = \text{�} \begin{array}{ll} 0, & (mm+ll=0) \\ \dfrac{dd}{mm+ll}, & (ccttottiiwwiisstt) \end{array}$$

For project 6, we will normalize the edit distance to 1.0:

$$dd_{nnnnnnnmm}(llbbcc, ttyyxx) = 1.0$$

$$dd_{nnnnnnnmm}(llbbcc, llbbcc) = 0.0$$

$$dd_{nnnnnnnmm}(ll, llbb) = 0.333$$

$$dd_{nnnnnnnmm}(llbb, bbcc) = 0.5$$

Self-study Project

$$DD_{ii,jj} = \min \begin{cases} DD_{ii-1,jj-1} + 0 & (ss_{jj} = tt_{ii}) & ccccppyy \\ DD_{ii-1,jj-1} + ff\left(ss_{jj}, tt_{ii}\right) & (ss_{jj} \neq tt_{ii}) & ssssbbsstt. \\ DD_{ii-1,jj} + 1 & & iillssttiitt \\ DD_{ii,jj-1} + 1 & & ddttllttttttt \end{cases}$$

$$DD_{ii,0} = ii, DD_{0,jj} = jj$$

$$dd = DD_{nn,mm}$$

$$dd_{nnnnnnmm} = \cdots$$

comparing texts (by string)

$$\boxed{ff\left(ss_{jj}, tt_{ii}\right) = 0.5 + dd_{nnnnnnmm}\left(ss_{jj}, tt_{ii}\right)}$$

comparing strings (by character)

$$\boxed{ff(ss_{jj}, tt)_{ii} = 2.0}$$

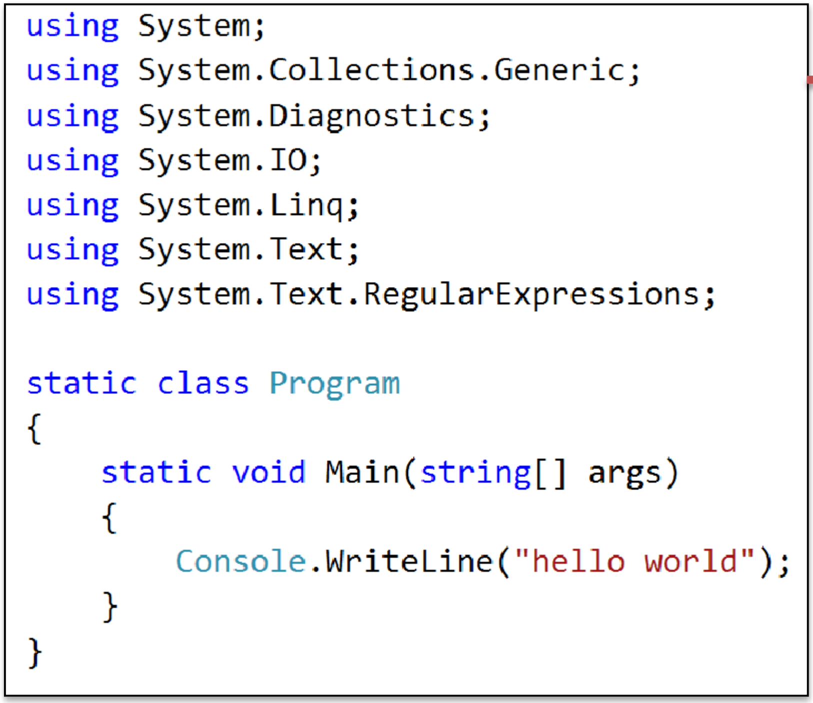
Self-study Project

- There are some nice opportunities to use elegant programming constructs in the self-study project
- We'll review some in the next slides:
  - Template functions
  - Jagged v. rectangular arrays
  - Lambda functions
- Specific techniques are not required for the project, but I encourage you to always be open to adding new techniques to your programming toolbox
- The more tools you have at your disposal, the more productive you'll be

Running C# examples

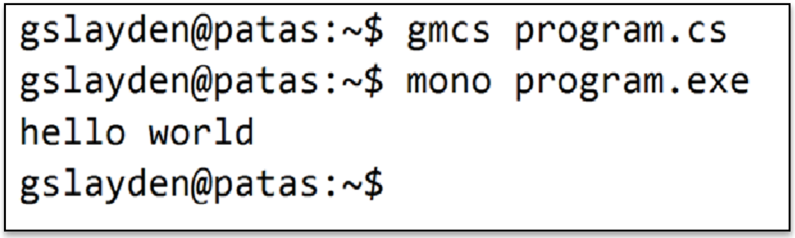
- This lecture will have some simple C# examples. To try them out on *patas/dryas*, you can adapt or play with the following skeleton file, `program.cs`

the system stuff that comes in handy



```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;

static class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("hello world");
    }
}
```

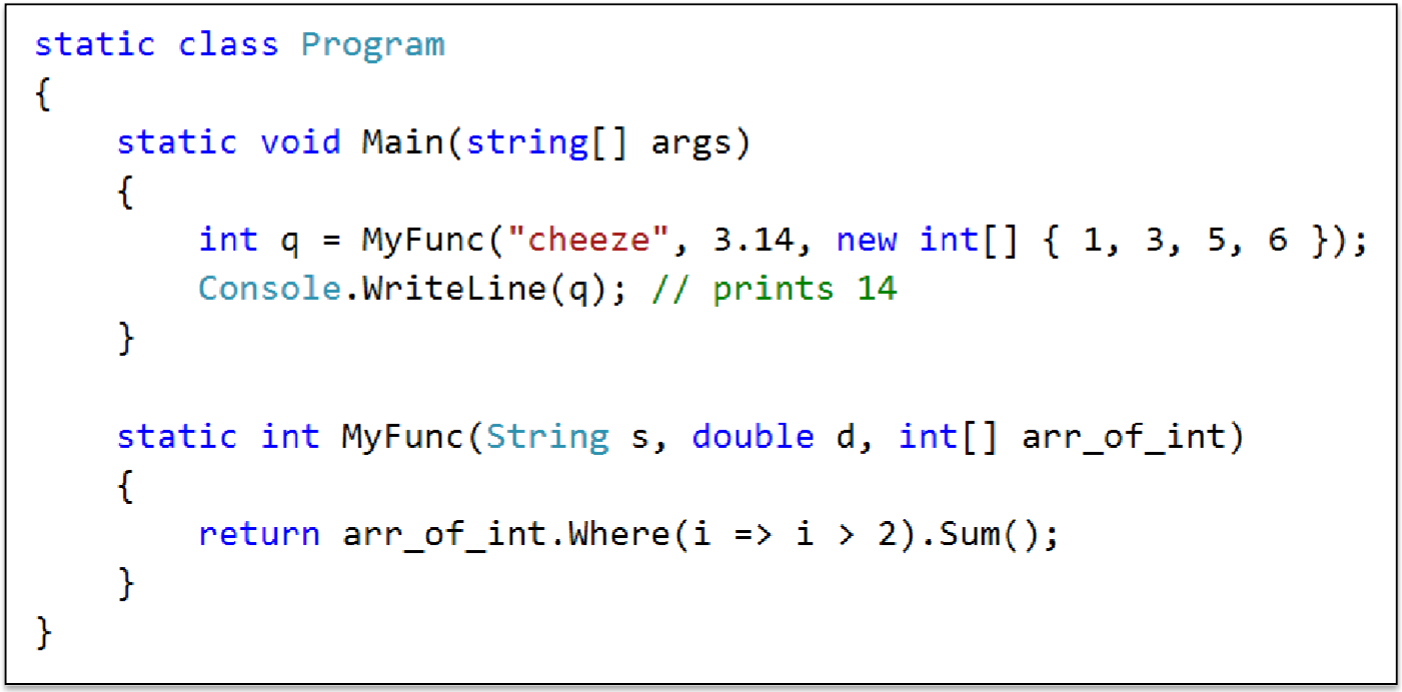
How to compile and run this C# program on patas/dryas



```
gslayden@patas:~$ gmcs program.cs
gslayden@patas:~$ mono program.exe
hello world
gslayden@patas:~$
```

Extending C# programs

- Like java, C# functions must be in classes
- If you don’t need to define your own object classes, just use static functions in a static class



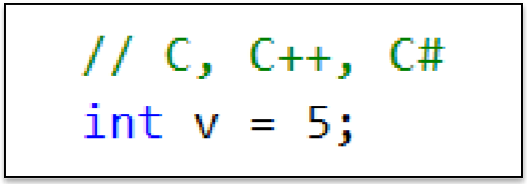
```
static class Program
{
    static void Main(string[] args)
    {
        int q = MyFunc("cheeze", 3.14, new int[] { 1, 3, 5, 6 });
        Console.WriteLine(q); // prints 14
    }

    static int MyFunc(String s, double d, int[] arr_of_int)
    {
        return arr_of_int.Where(i => i > 2).Sum();
    }
}
```

Computer science: types

- The idea of types is useful in all programming languages.
  - byte : 8 bits of arbitrary data
  - short : an integer that fits in (e.g.) 16-bits
  - integer : an integer that fits in (e.g.) 32-bits
  - long : an integer that fits in (e.g.) 64-bits
  - double : a 64-bit floating point number
  - string : a sequence of ll characters
  - float : a 32-bit floating point number
  - boolean : a true-or-false value
  - MyClass : a composed, user-defined entity

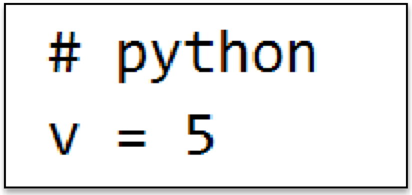
Declaring variables

- Some languages require you to declare the type of a variable before you use it



```
// C, C++, C#
int v = 5;
```

- Others don’t



```
# python
v = 5
```

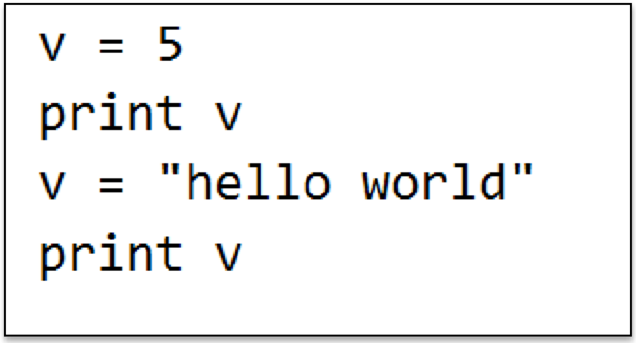
Strong (static) v. dynamic typing

- Programming languages can be strongly typed (C#, java) or dynamically typed (Python, Basic, Javascript, Perl)
- Static (strong) type enforcement allows more errors to be caught before running the program, because compilers and editing tools can flag inconsistent usages which are probably programming errors
- In reality, there is a spectrum of type strength. Polymorphism in strongly-typed languages is a controlled form of dynamic typing

Strong typing is often considered a productivity gain, because it uncovers conceptual errors earlier in the development process.

example

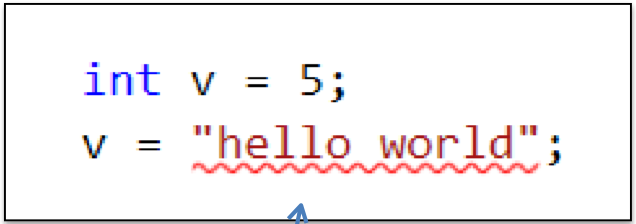
Python:



```
v = 5
print v
v = "hello world"
print v
```

✓

C#:



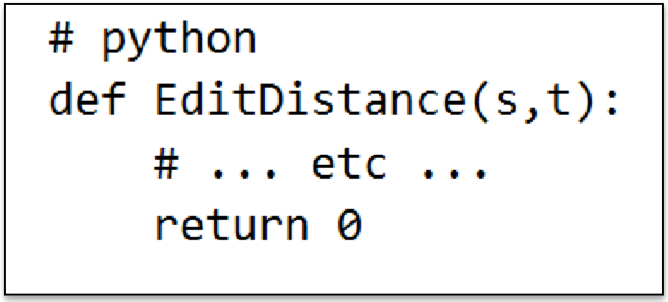
```
int v = 5;
v = "hello world";
```

✗

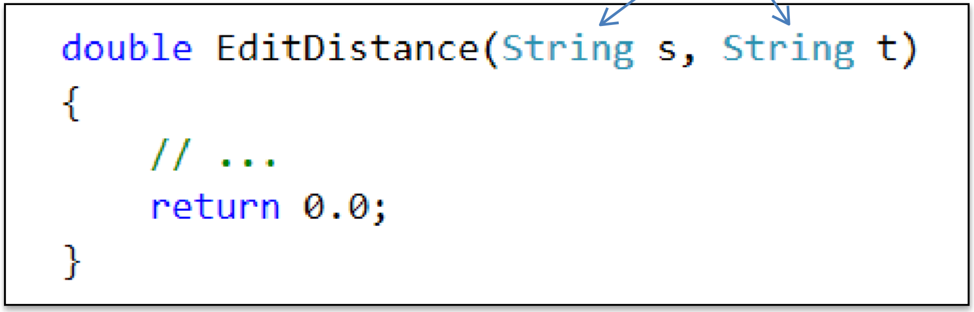
The editor has already flagged this as an error, as soon as you wrote it

Flexible functions

- For Project 6, we would like a function that operates on a sequence of elements—which could be of any type
- This is no problem in a dynamically typed language, but in C#, would we have to commit to a type?



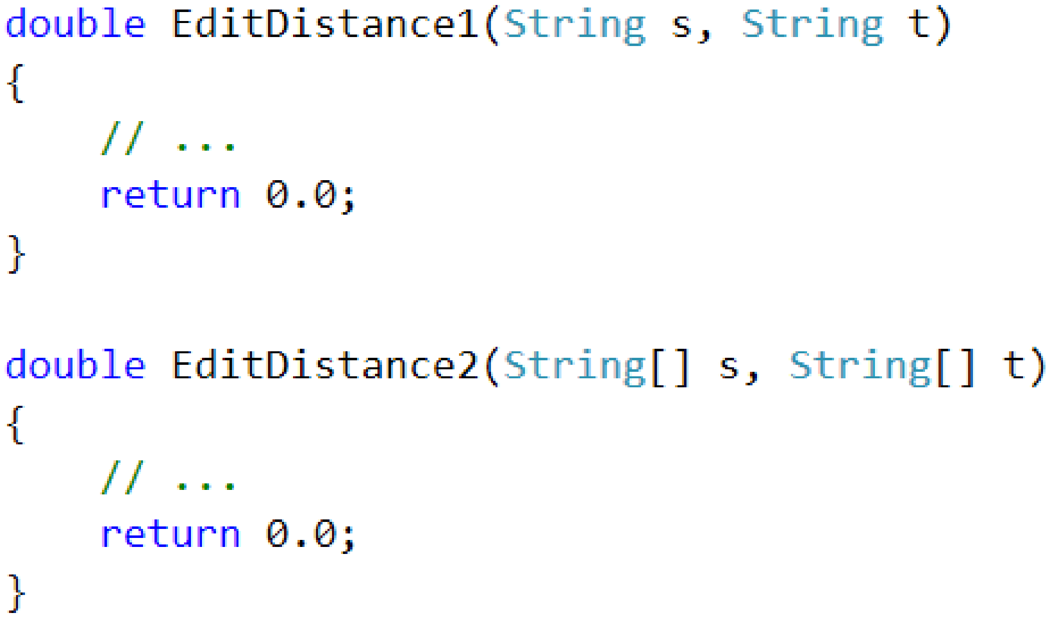
```
# python
def EditDistance(s,t):
    # ... etc ...
    return 0
```



```
double EditDistance(String s, String t)
{
    // ...
    return 0.0;
}
```

The easy solution is to repeat the entire function, once for each different type you anticipate. Problems:

- You may not anticipate future uses with other types
- The code is duplicated, which invites bugs



```
double EditDistance1(String s, String t)
{
    // ...
    return 0.0;
}

double EditDistance2(String[] s, String[] t)
{
    // ...
    return 0.0;
}
```

Programming with templates

- Fortunately, strongly-typed languages have features that allow for this
  - You can specify exactly which arguments of a function (or parts of an entire class) are type-flexible
  - In C#, you can apply special *constraints* on the allowable types, which *expands* the things you can do with the types in the function
  - The language and environment automatically create instances of the function (or object) upon demand, even for unforeseen types.

Lambda expressions

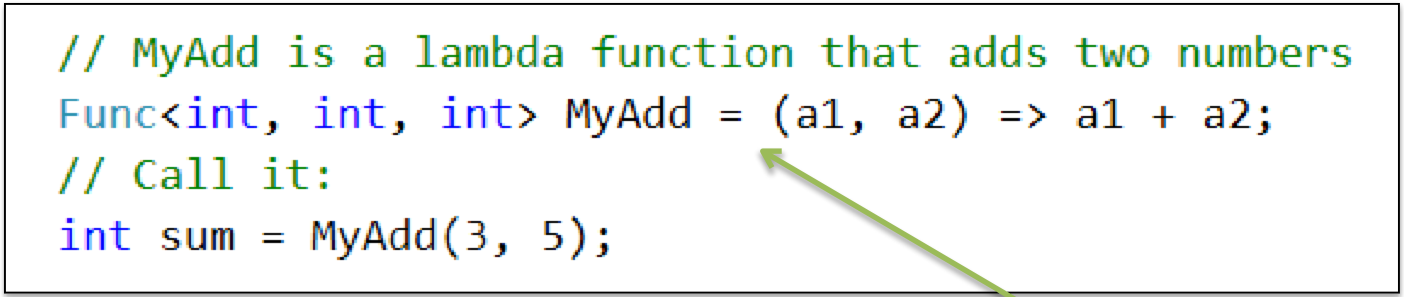
- A lambda expression is a portable, possibly anonymous (unnamed) snippet of code that you can store, refer to and carry and pass around just like any other data object
- It's an elegant way for callers to customize some aspect of a function's behavior
  - This is exactly what the EditDistance function requires:
  - A way to allow callers to arbitrarily customize the substitution cost function

What’s the ‘type’ of a lambda function?

- Like (most) objects in C#, a lambda function must be strongly typed
- This means defining the exact types expected for:
  - One or more input parameters
  - The return value (if any)
- Fortunately, the system libraries provide a template class, so you can specify the parameter types and return value type for any strongly-typed lambda function you need

Func<T1,T2,…,TReturn>

Use this system-defined template class to create lambda functions that have a return value

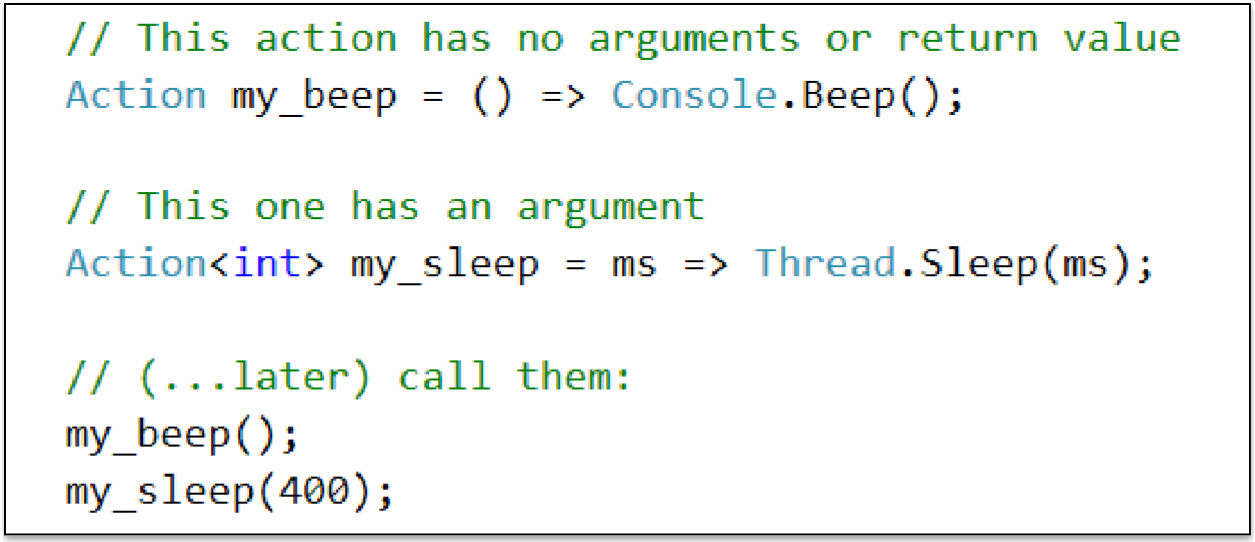


```
// MyAdd is a lambda function that adds two numbers
Func<int, int, int> MyAdd = (a1, a2) => a1 + a2;
// Call it:
int sum = MyAdd(3, 5);
```

No adding happens here at this point; we’re just declaring the function

Action<T1,T2,…>

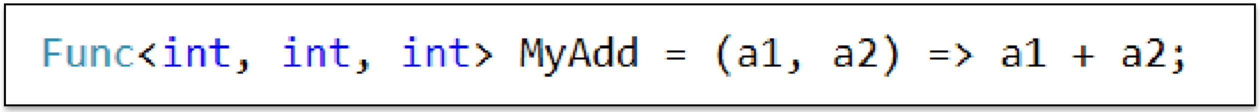
`Action<…>` is for lambda functions that do not return a value



```
// This action has no arguments or return value
Action my_beep = () => Console.Beep();

// This one has an argument
Action<int> my_sleep = ms => Thread.Sleep(ms);

// (...later) call them:
my_beep();
my_sleep(400);
```

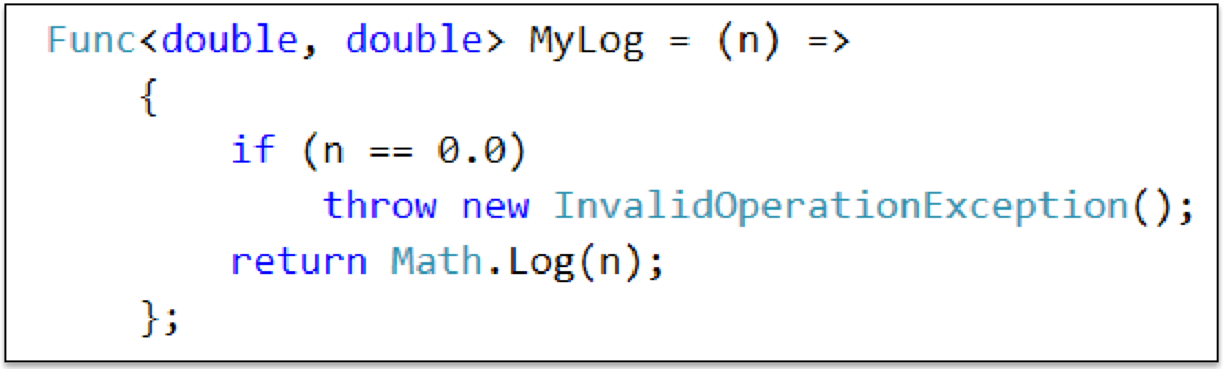
Two syntaxes

- There are two syntaxes for lambda functions in C#. If it’s a short function, you can do it as shown on the previous slides:



```
Func<int, int, int> MyAdd = (a1, a2) => a1 + a2;
```

- If it’s longer than one line, you might prefer to write it as shown below instead. If you use this curly brace syntax, you have to use the `return` keyword.



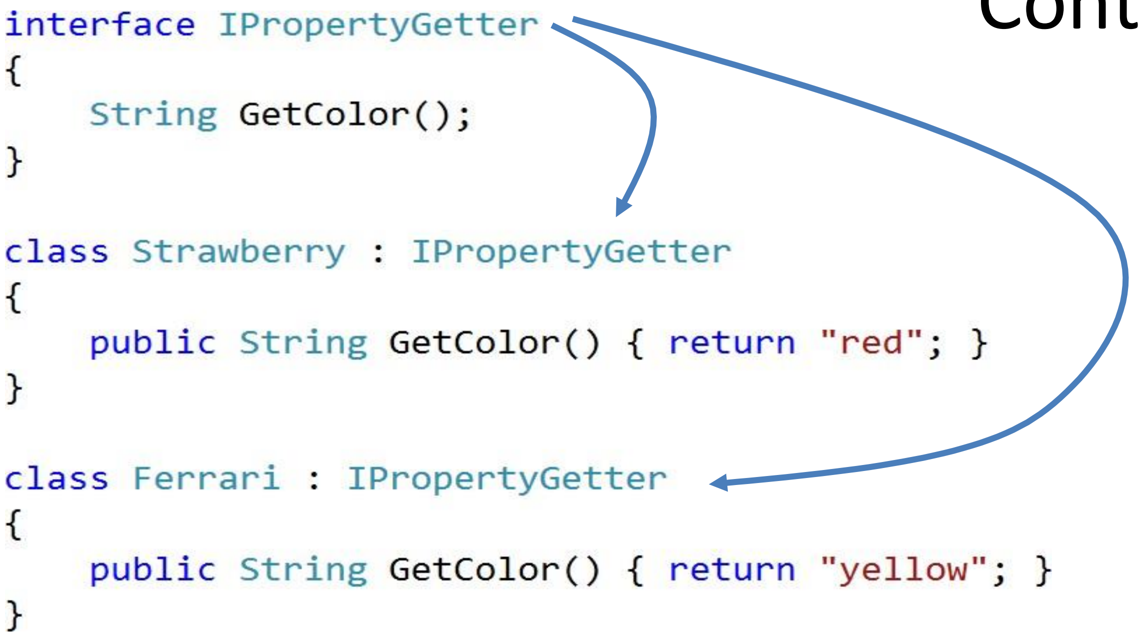
```
Func<double, double> MyLog = (n) =>
    {
        if (n == 0.0)
            throw new InvalidOperationException();
        return Math.Log(n);
    };
```

Interfaces

An interface is a named set of zero or more function signatures with no implementation(s)

- To implement an interface, a class defines a matching implementation for every function in the interface
- Interfaces are sometimes described as contracts
- You can define and use a reference to an interface just like any other object reference

Contrived Example



```
interface IPropertyGetter
{
    String GetColor();
}

class Strawberry : IPropertyGetter
{
    public String GetColor() { return "red"; }
}

class Ferrari : IPropertyGetter
{
    public String GetColor() { return "yellow"; }
}
```

- This looks like C++ class inheritance
  - yes, but it’s more ad-hoc
  - C# classes can have single inheritance of other classes, and multiple inheritance of interfaces
  - Interfaces can inherit from other interfaces (not shown)

Case Study
interfaces and templates:
IEnumerable<T>

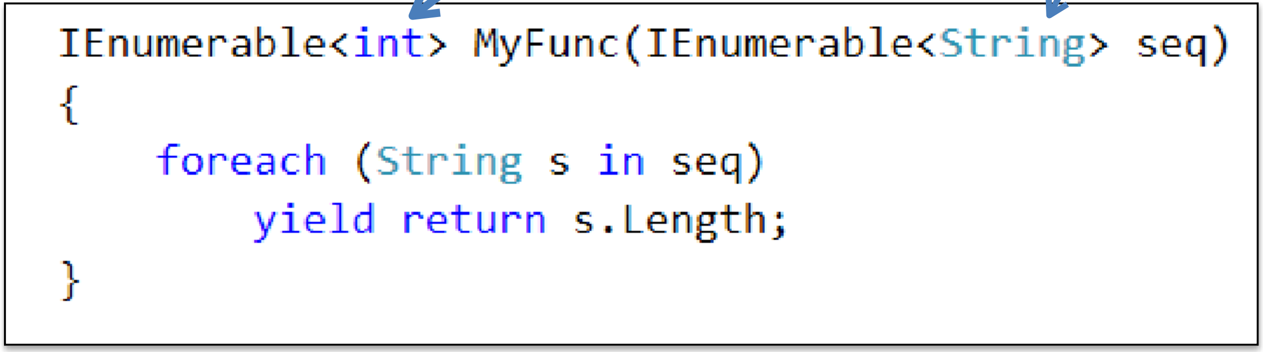
`IEnumerable<T>` is one of many system-defined interfaces that a class can elect to implement

IEnumerable<T>

- This is one of the simplest interfaces defined in the BCL (base class libraries)
- This interface provides just one thing: a way to iterate over elements of type T
- All of the system arrays, collections, dictionaries, hash sets, etc. implement `IEnumerable<T>`
  - Implementing IEnumerable<T> on your own classes can be very useful, but you don't need to worry about that
  - For now, what's important is that you get to use it, because it's available on all of the system collections

IEnumerable<T>

- This is an interface (ad hoc grouping of functions) that allows for enumeration of (strongly-typed) objects of type T.
- So we can lazily transform a sequence of strings into a sequence of their integer lengths:



```
IEnumerable<int> MyFunc(IEnumerable<String> seq)
{
    foreach (String s in seq)
        yield return s.Length;
}
```

IEnumerator<T>

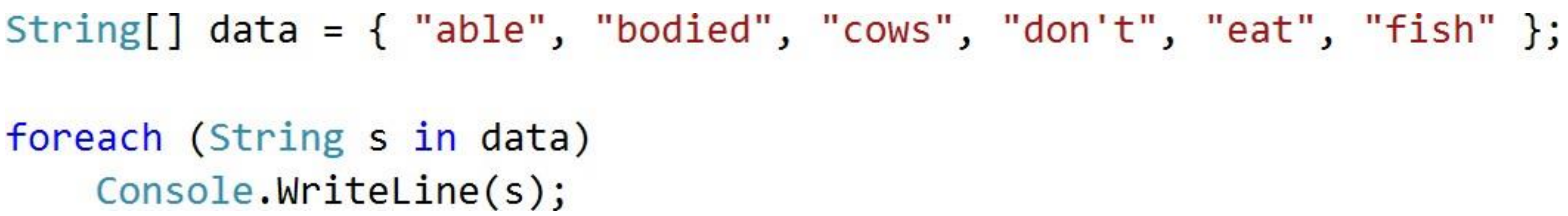
- IEnumer**able**<T> has only one function, which allows a caller or caller(s) to obtain an *enumerator* object which is able to iterate over elements
  - The actual enumerator object is an object that implements a different interface, called IEnumera**tor**<T>
  - This "factory" design allows a caller to initiate and maintain several simultaneous iterations if needed
  - The enumerator object, IEnumerator<T> can only:
    - Get the current element
    - Move to the next element
    - Tell you if you've reached the end
  - Note: There's no count
    - ICollection inherits from IEnumerable to provide this

IEnumerable, yield, and deferred execution

- Before describing the trie data structure, let’s look at iterators which enumerate a sequence of elements

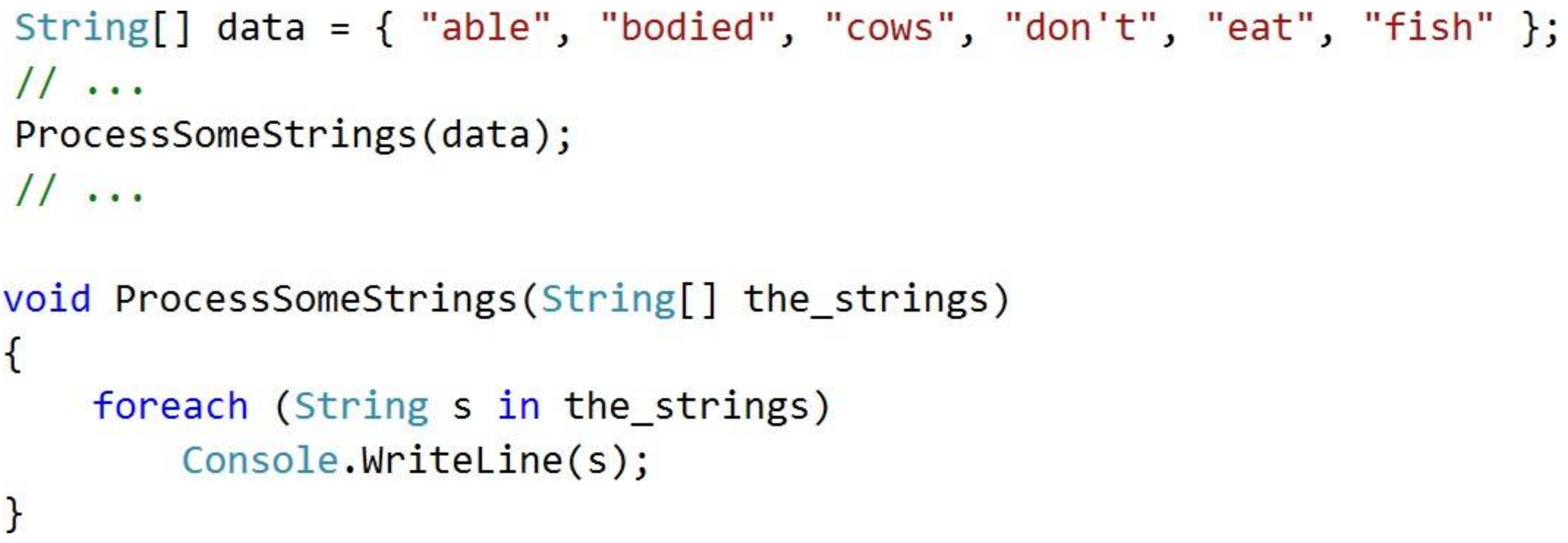
Examples in C#. If you use another language, it will be instructive to think about how to adapt the solutions to your language

- Enumeration is obvious when the data is at hand and you want to use it all:



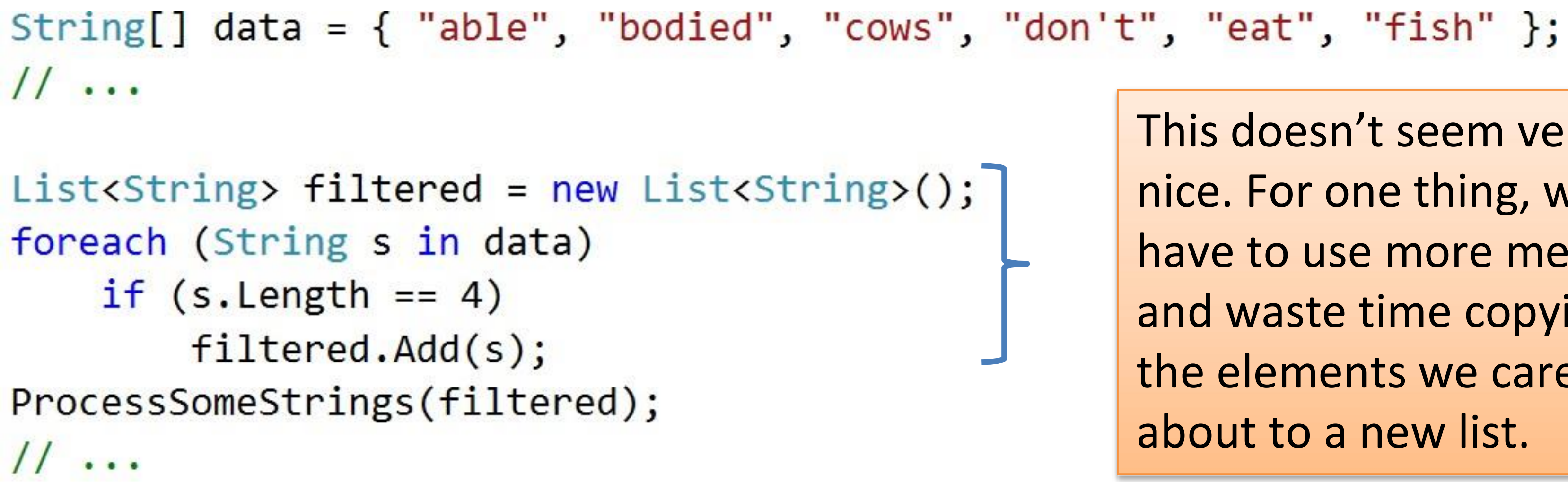
```
String[] data = { "able", "bodied", "cows", "don't", "eat", "fish" };

foreach (String s in data)
    Console.WriteLine(s);
```

We can pass (a reference to) the array around too, no problem

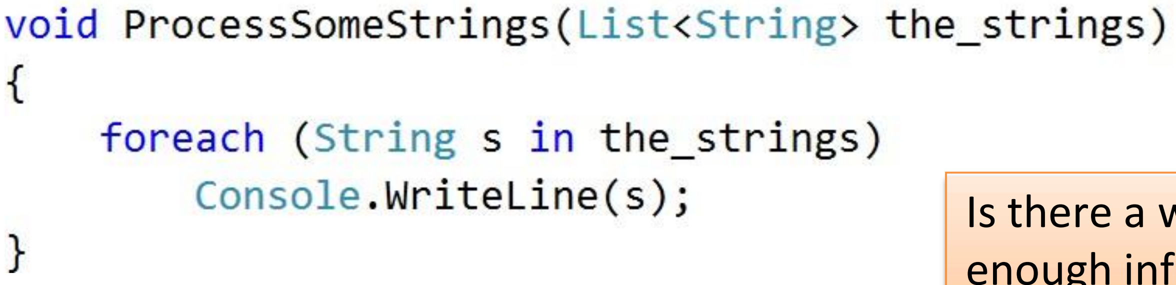


```
String[] data = { "able", "bodied", "cows", "don't", "eat", "fish" };
// ...
ProcessSomeStrings(data);
// ...

void ProcessSomeStrings(String[] the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

What if we only want to "process" the four-letter words?



```
String[] data = { "able", "bodied", "cows", "don't", "eat", "fish" };
// ...

List<String> filtered = new List<String>();
foreach (String s in data)
    if (s.Length == 4)
        filtered.Add(s);
ProcessSomeStrings(filtered);
// ...
```

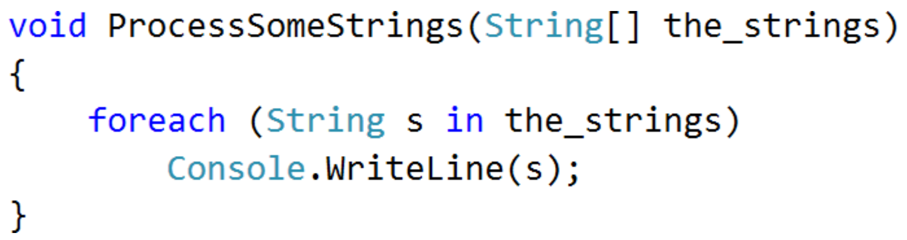


```
void ProcessSomeStrings(List<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

This doesn't seem very nice. For one thing, we have to use more memory and waste time copying the elements we care about to a new list.

Is there a way to pass this function enough information to filter the *original list* itself, where it lies?

- Remember the non-filtered example for a second

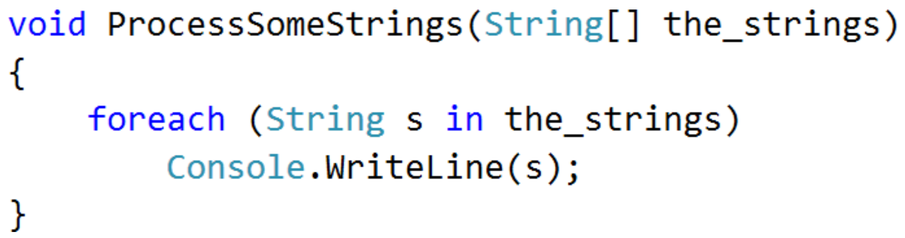


```
void ProcessSomeStrings(String[] the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

- The processing function doesn't really care about the fact that the data is in an array
- This violates an important programming maxim:

A flexible interface *demands the least* and *provides the most*:

- *Inputs* are as general as possible (allowing clients to supply any level of specificity, i.e. be lazy)
- *Outputs* are as specific as possible (allowing clients to capitalize on work products, i.e. be lazy).



```
void ProcessSomeStrings(String[] the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

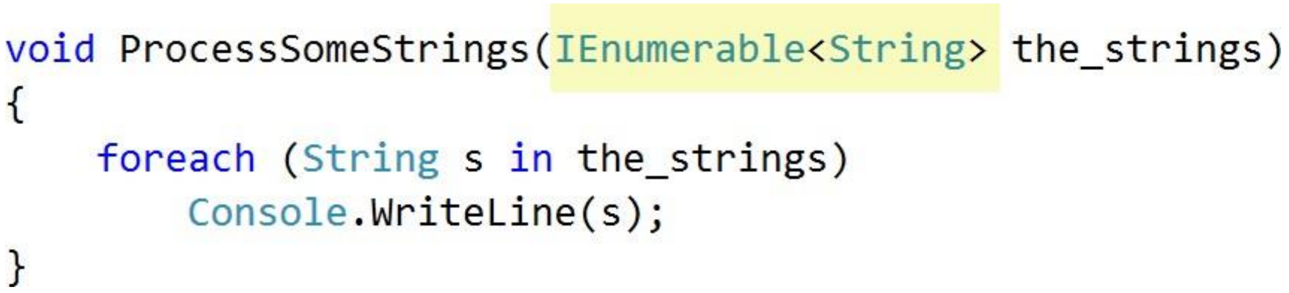
The extra (unused) demands this function is making by asking for String[]:

- That the strings all be in memory at the same time
- That the strings be randomly accessible by an index
- That the number of strings be known and fixed before the function starts

- To modify this to comply with the maxim, we first ask:
- Q: What is the absolute minimum that this function actually needs to accomplish it’s work?
- Answer: a way to iterate strings

Interfaces as function arguments

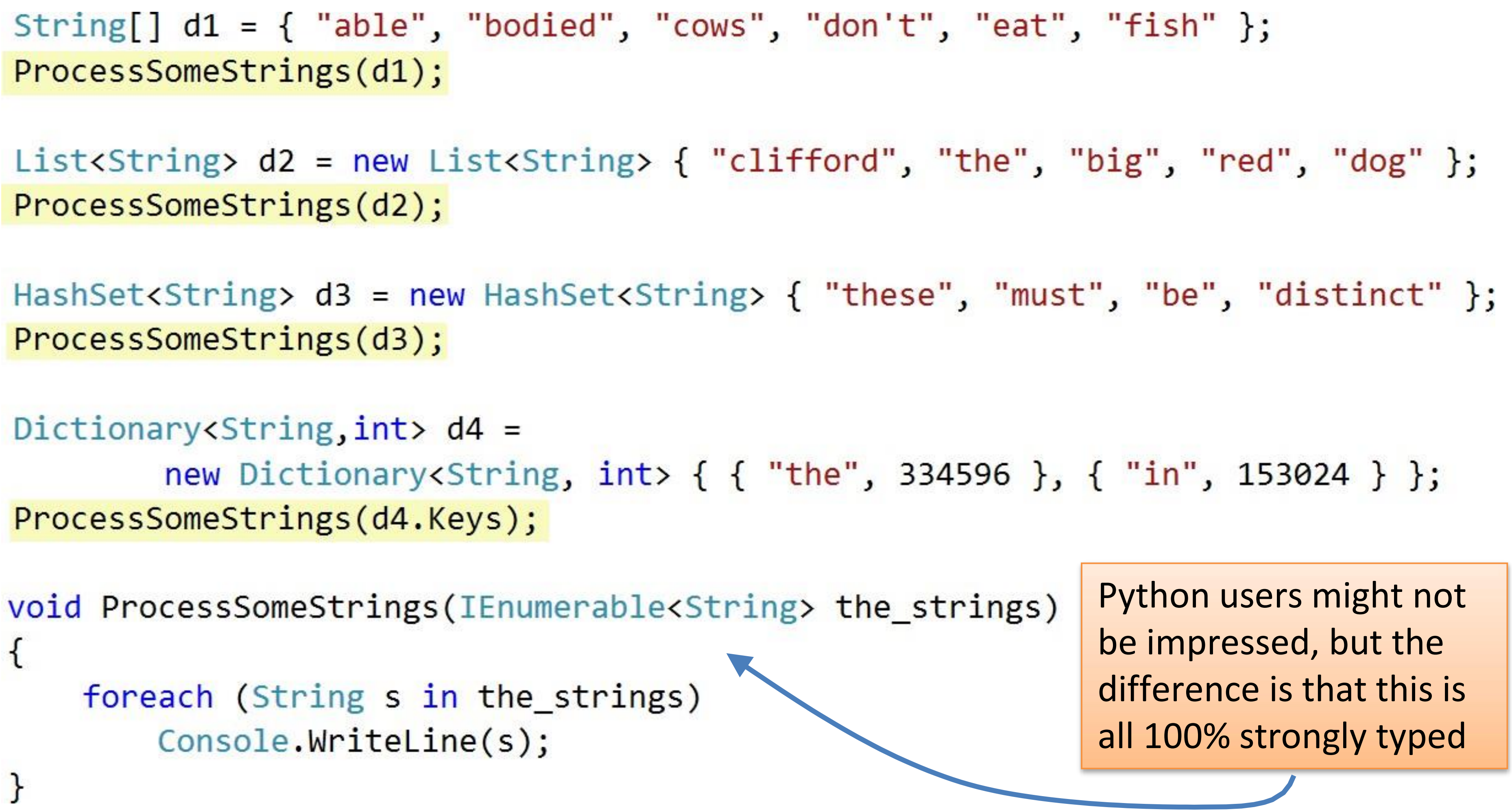
- Using interfaces as function arguments allows you to require the absolute minimum functionality the function actually needs
- In this way, the ad-hoc nature of interfaces allows us to comply with the maxim



```
void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

Now, this function is exposing the weakest (most general) requirement possible for the processing it has to do. This provides more flexibility to callers since they can choose whatever level of specificity is convenient. The function can be used in the widest possible variety of situations.

Example



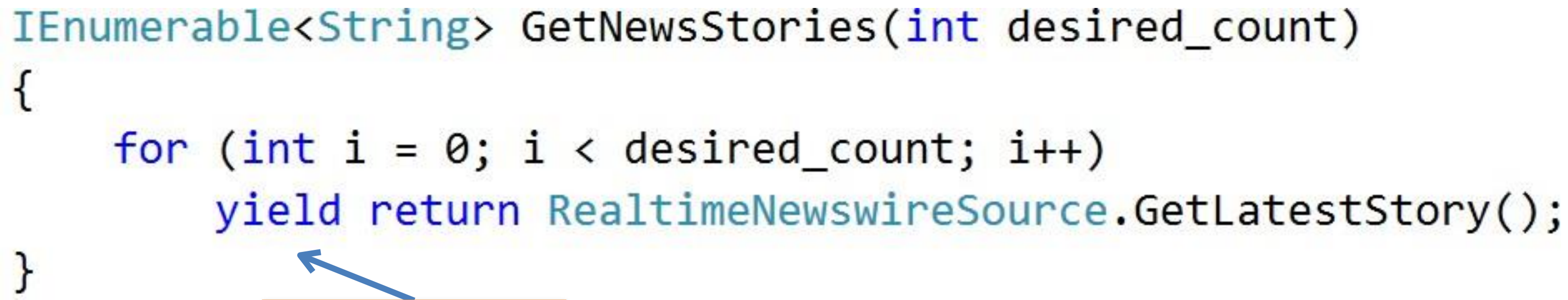
```
String[] d1 = { "able", "bodied", "cows", "don't", "eat", "fish" };
ProcessSomeStrings(d1);

List<String> d2 = new List<String> { "clifford", "the", "big", "red", "dog" };
ProcessSomeStrings(d2);

HashSet<String> d3 = new HashSet<String> { "these", "must", "be", "distinct" };
ProcessSomeStrings(d3);

Dictionary<String,int> d4 =
        new Dictionary<String, int> { { "the", 334596 }, { "in", 153024 } };
ProcessSomeStrings(d4.Keys);

void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

Python users might not be impressed, but the difference is that this is all 100% strongly typed

Iteration is efficient

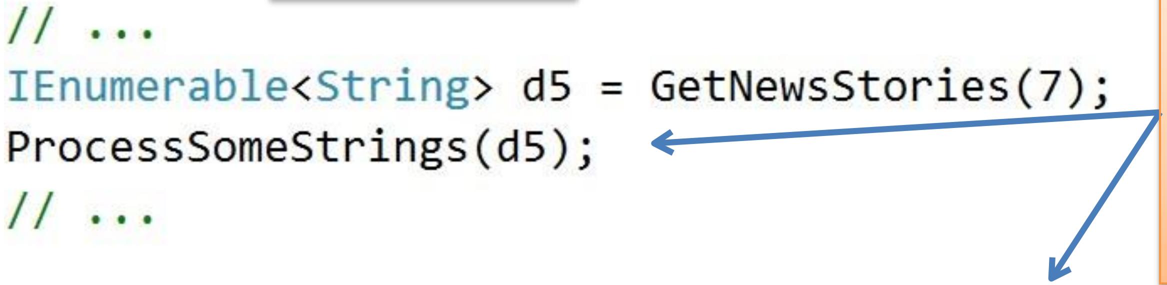
- That’s cool, IEnumerable<T> lets a function not care about where a sequence of elements is coming from
  - We don’t copy the elements around
  - Iterators let us access elements right from their source
- All of those examples iterate over elements that already exist somewhere
- Is there a way to iterate over data that’s generated on-the-fly, doesn’t exist yet, or is never persisted at all?
- Yes!

Iterating over on-the-fly data



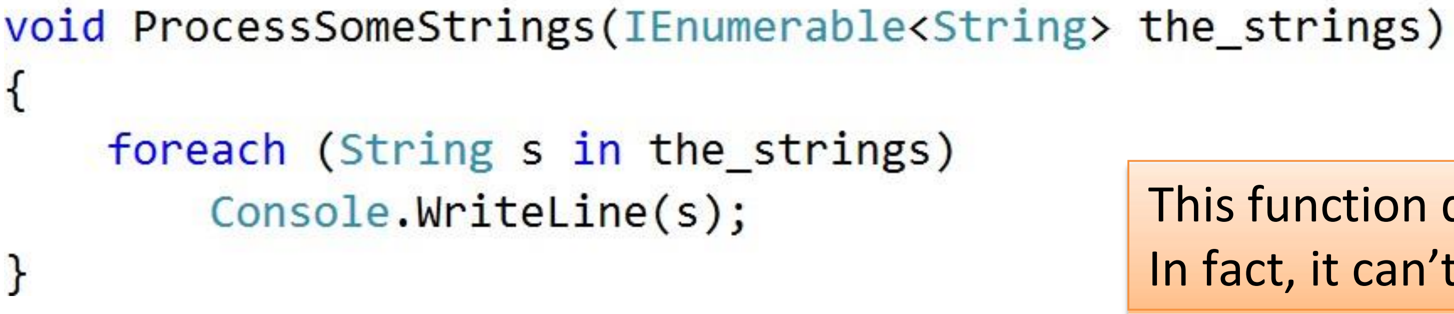
```
IEnumerable<String> GetNewsStories(int desired_count)
{
    for (int i = 0; i < desired_count; i++)
        yield return RealtimeNewswireSource.GetLatestStory();
}
```

see next slide



```
// ...
IEnumerable<String> d5 = GetNewsStories(7);
ProcessSomeStrings(d5);
// ...
```

This is exactly the same as before, but this time there's no "collection" of elements sitting anywhere



```
void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

This function doesn't care. In fact, it can't even tell.

yield keyword

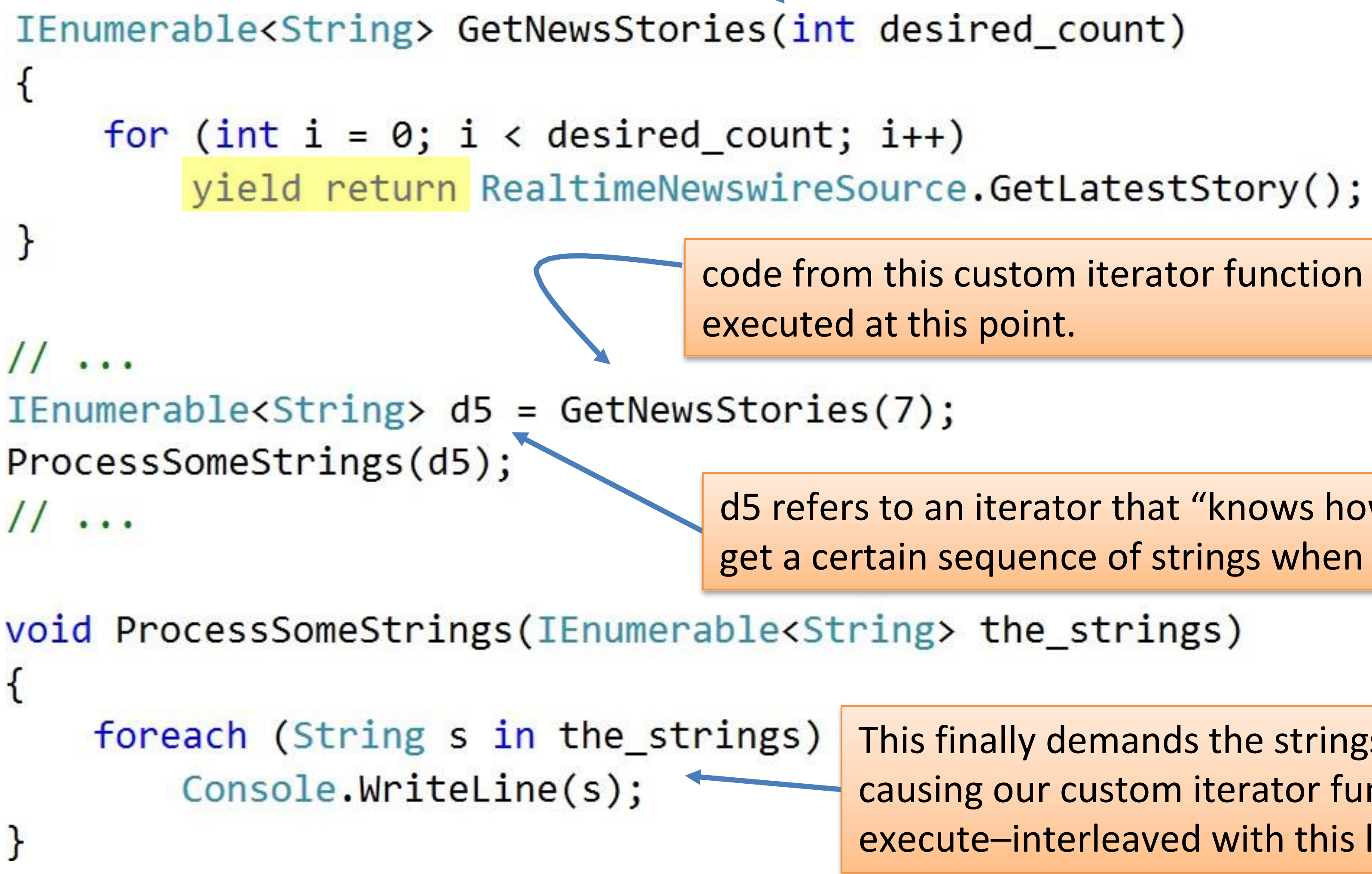
- The `yield` keyword makes it easy to define your own custom iterator functions
- Any function that contains the `yield` keyword becomes special
  - It must be declared as returning an IEnumerable<T>
  - Deferred execution means that the function's body is not necessarily invoked when you "call" it
  - It must deliver zero or more elements of type T using:

    ```
    yield return t;
    ```

  - Sometime later, control may continue immediately after this statement to allow you to yield additional elements
  - It may signal the end of the sequence by using:

    ```
    yield break;
    ```

Custom iterator function example



```
IEnumerable<String> GetNewsStories(int desired_count)
{
    for (int i = 0; i < desired_count; i++)
        yield return RealtimeNewswireSource.GetLatestStory();
}


// ...
IEnumerable<String> d5 = GetNewsStories(7);
ProcessSomeStrings(d5);
// ...


void ProcessSomeStrings(IEnumerable<String> the_strings)
{
    foreach (String s in the_strings)
        Console.WriteLine(s);
}
```

code from this custom iterator function is *not* executed at this point.

d5 refers to an iterator that "knows how" to get a certain sequence of strings when asked

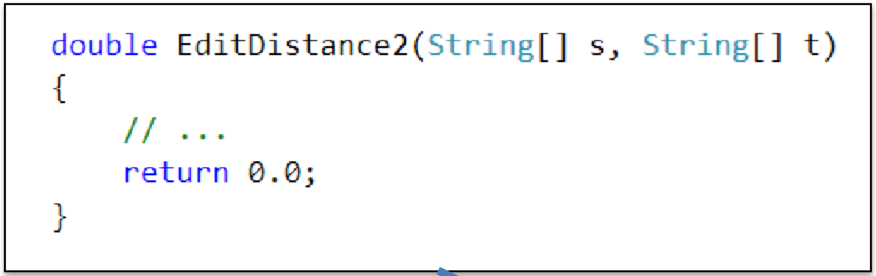
This finally demands the strings, causing our custom iterator function to execute–interleaved with this loop!

…end of the case study

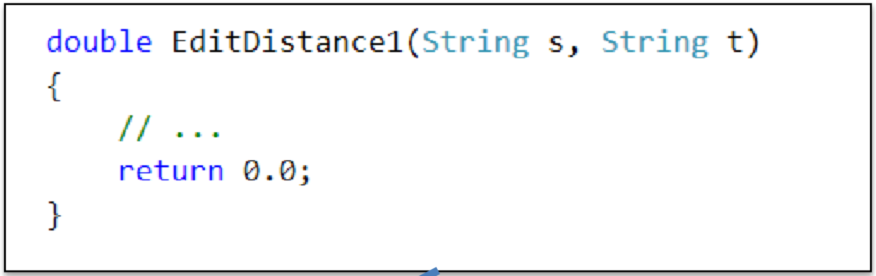
now back to Project 6…

Function templates

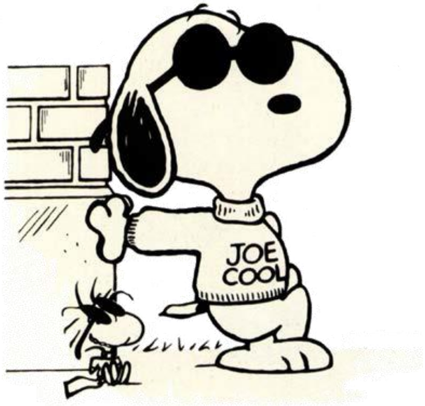
- You can use templates to allow your strongly-typed functions to be flexible.

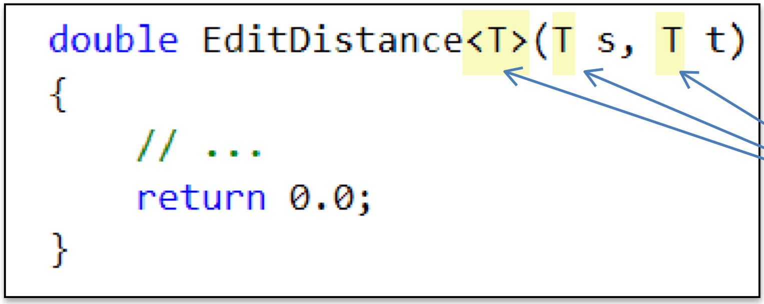


```
double EditDistance2(String[] s, String[] t)
{
    // ...
    return 0.0;
}
```



```
double EditDistance1(String s, String t)
{
    // ...
    return 0.0;
}
```





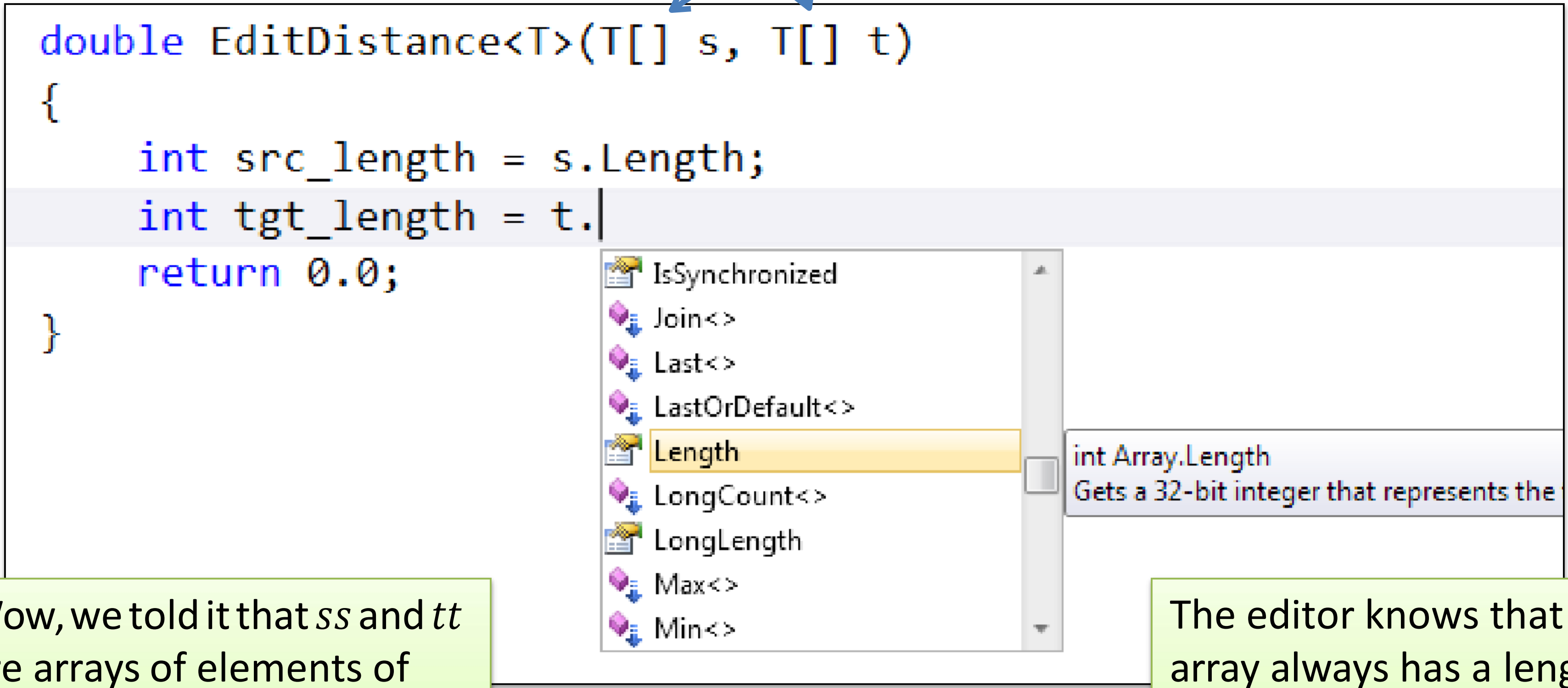
```
double EditDistance<T>(T s, T t)
{
    // ...
    return 0.0;
}
```

T can be any identifier name. It's convention to use upper case letters

Useful?

- Oops, it’s going to be hard to use s and t for anything in your function body, though because:
  - the compiler can’t infer much about it, except that…
  - …it inherits from type “object”
  - The language is still strongly typed, so inside your function you won’t be allowed to do anything that is more specific than what you can do with “object”
  - Actually, you could cast them at runtime to some specific type, but this means you’ve lost your type safety and you’re vulnerable to runtime errors (like a dynamically typed language)

We know that our function always deals with sequences. Let’s declare that.



```
double EditDistance<T>(T[] s, T[] t)
{
    int src_length = s.Length;
    int tgt_length = t.
    return 0.0;
}
```

int Array.Length
Gets a 32-bit integer that represents the

Wow, we told it that *ss* and *tt* are arrays of elements of type T, and strong typing is back!

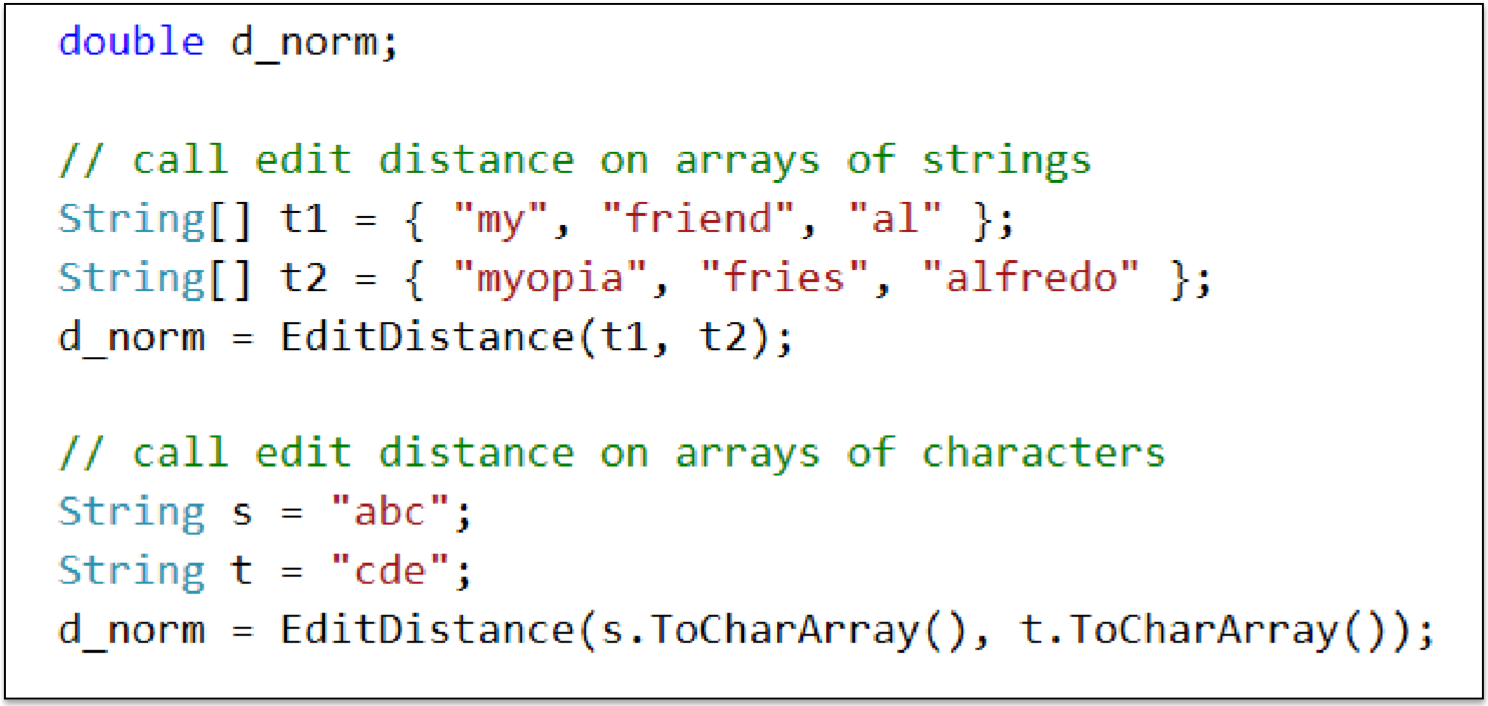
The editor knows that an array always has a length property.

A note for C++ users

- Templates are a first-class feature of the mono/CLR runtime *environment*, not the C# language per se.
- They are not fully resolved at compile-time like C++ templates
- In certain cases, a new version of your template function or class can be generated by the runtime environment, specialized for a type that may not have even existed when you wrote and compiled your program
  - This happens without needing the source code to your program, or re-compiling from the source code

Calling the template function

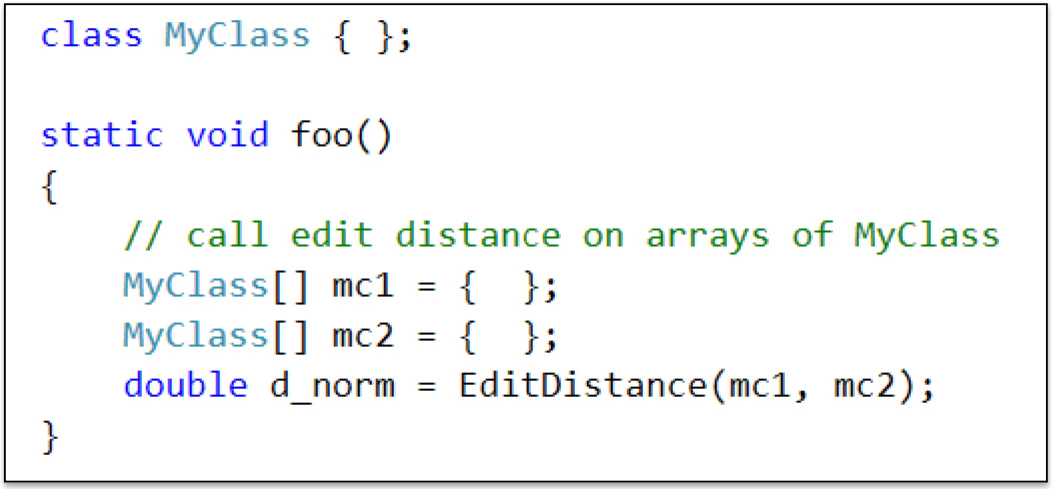
- Now we can call the function with an array of any type. The compiler figures out what type T is automatically



```
double d_norm;

// call edit distance on arrays of strings
String[] t1 = { "my", "friend", "al" };
String[] t2 = { "myopia", "fries", "alfredo" };
d_norm = EditDistance(t1, t2);

// call edit distance on arrays of characters
String s = "abc";
String t = "cde";
d_norm = EditDistance(s.ToCharArray(), t.ToCharArray());
```

Template amok?

- As it currently stands, we can also call our function with a an array of any other type of element(s)



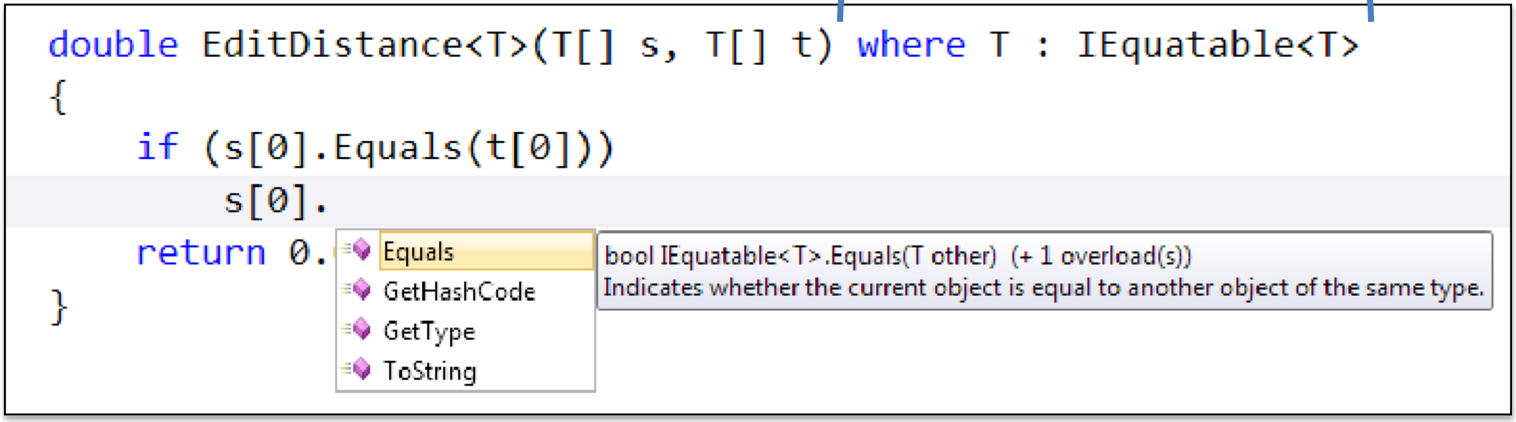
```
class MyClass { };

static void foo()
{
    // call edit distance on arrays of MyClass
    MyClass[] mc1 = {  };
    MyClass[] mc2 = {  };
    double d_norm = EditDistance(mc1, mc2);
}
```

- We may or may not want this. Rather than forbid specific types, we can declare the minimum set of constraints that EditDistance(…) actually needs in order to operate.

Template constraints

- You can restrict T in various ways to allow you to do more with objects of type T in the template function
- For the EditDistance function, it is useful to require that objects of type T be equatable

"T must be a type that implements the function Equals<T>(T t) which tests the equality of two objects of type T."



```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    if (s[0].Equals(t[0]))
        s[0].
    return 0.
}
```

bool IEquatable<T>.Equals(T other) (+ 1 overload(s))
Indicates whether the current object is equal to another object of the same type.

- After adding the constraint, the editor (and compiler) immediately know(s) that elements of *ss* and *tt* can be tested for equality

Can’t you just use == ?

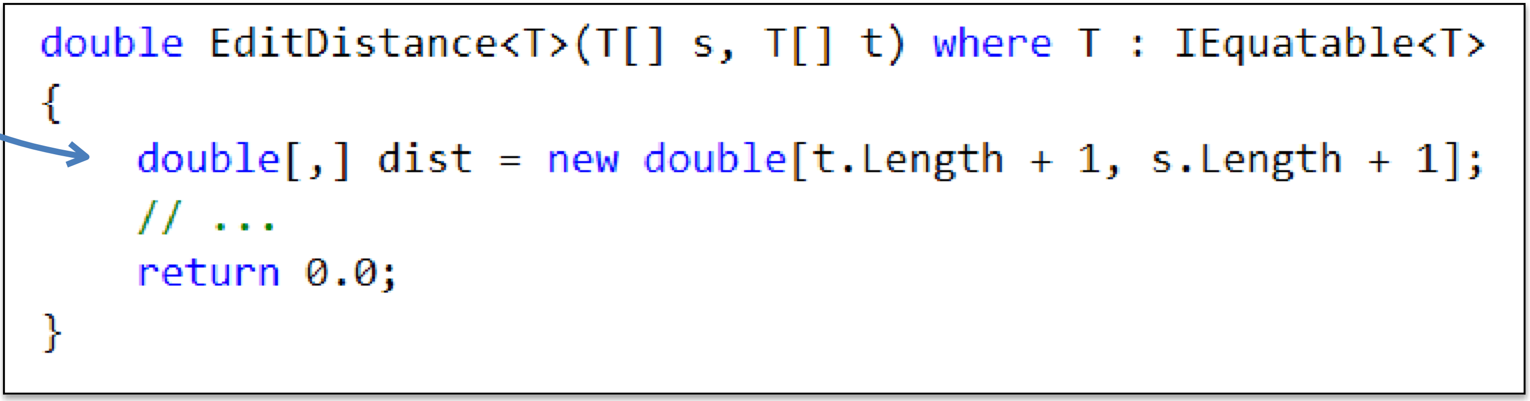
- On the previous slide, why couldn’t we just have compared objects of type T using ‘==‘ ?
- Because C# supports user-defined value types, which do not automatically implement the == operator
  - That’s because value types usually prefer to provide bitwise comparison semantics, as opposed to reference equality
- Since we didn’t constrain our template function to *exclude* value types (by saying `where T : class`), we can’t use that operator

Some details

- By default, *reference types* support *reference equality* via the == operator.
- However, `String` (for example) is one *reference type* which provides *value comparison semantics* instead.
- This is what you’d want and expect: “Felicia” == “Felicia” should be true even if the strings happen to be allocated in different places.

Jagged v. rectangular arrays

- Many languages support jagged versus rectangular arrays
- For project 6, you should use a rectangular array, if available
- This is how to define a rectangular array in C#



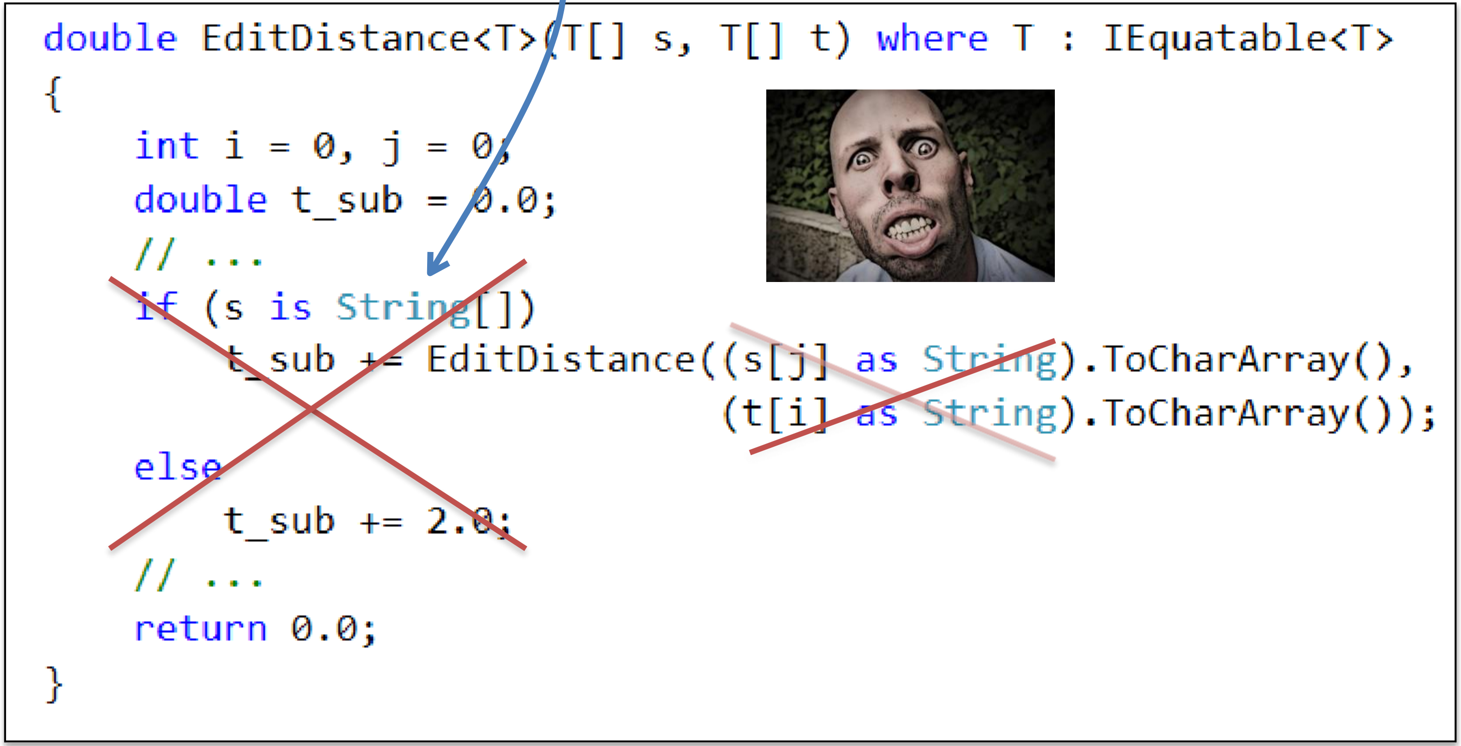
```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    double[,] dist = new double[t.Length + 1, s.Length + 1];
    // ...
    return 0.0;
}
```

Implementing the adjustable substitution cost function

- Lastly, the EditDistance function needs to use a different substitution cost function depending on whether you're doing the outer calculation (between the two texts) or inner calculation (between two lines of text)

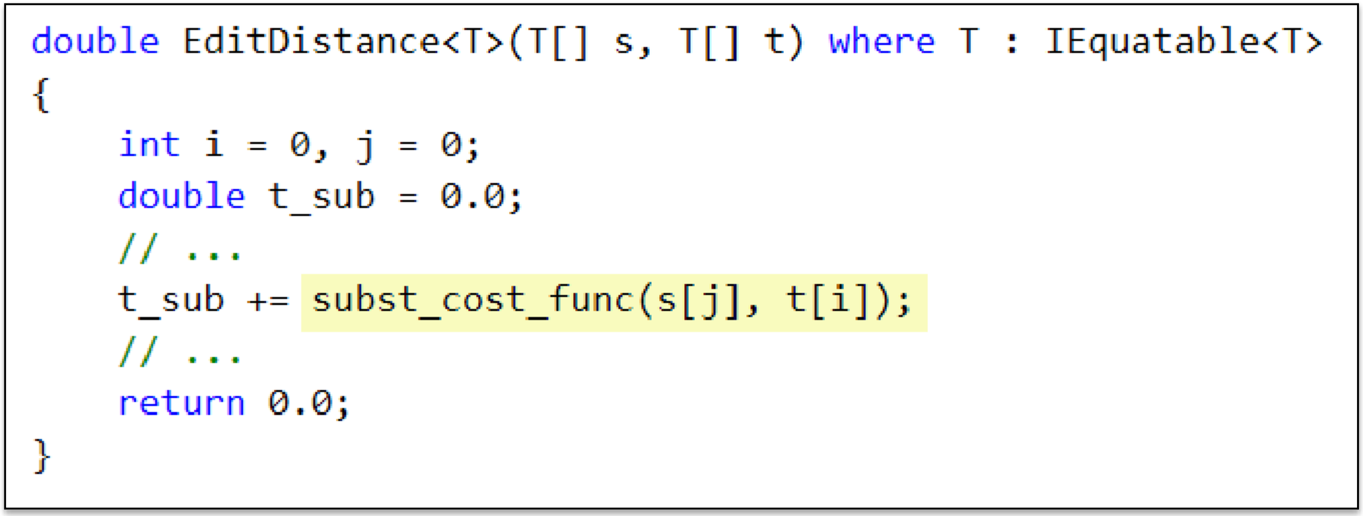
- If you've created duplicate versions of the function (not using a template), then you can just hard-code the appropriate cost function

However, if you liked the template idea so far and now you have a nice, type-agile function, I'm sure you wouldn't want to ruin it like this:



```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    int i = 0, j = 0;
    double t_sub = 0.0;
    // ...
    ~~if (s is String[])~~
        ~~t_sub += EditDistance((s[j]~~ ~~as String~~~~).ToCharArray(),~~
                                    ~~(t[i]~~ ~~as String~~~~).ToCharArray());~~
    ~~else~~
        ~~t_sub += 2.0;~~
    // ...
    return 0.0;
}
```

Instead, you’d like to do something like this:



```
double EditDistance<T>(T[] s, T[] t) where T : IEquatable<T>
{
    int i = 0, j = 0;
    double t_sub = 0.0;
    // ...
    t_sub += subst_cost_func(s[j], t[i]);
    // ...
    return 0.0;
}
```

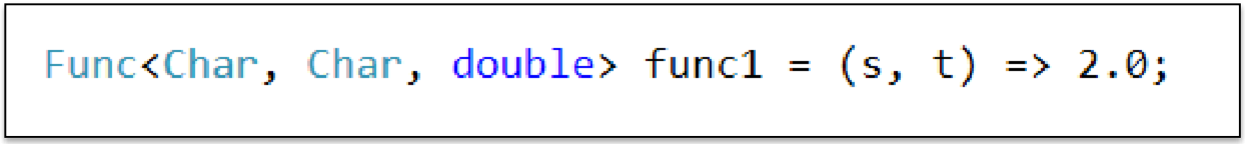
Hmm, how do we declare this function in terms of T, though?

This is a great place to use a lambda function

Self-study project

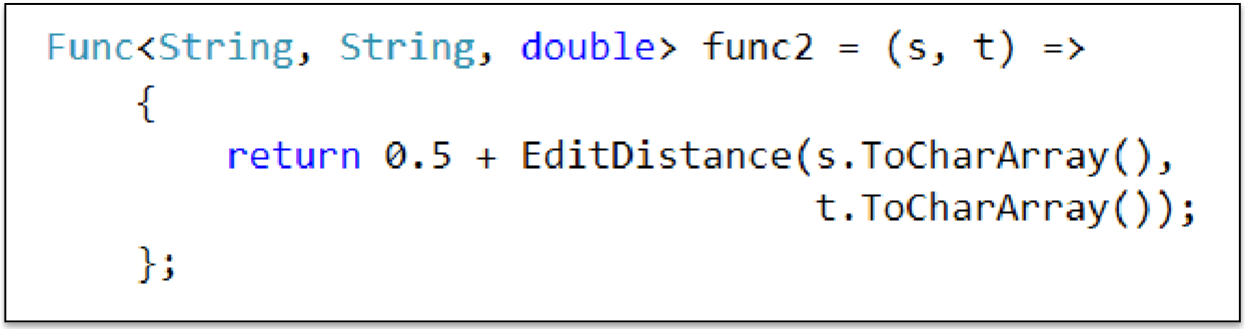
- Here are the two different substitution cost functions that we’d like to “pass in” to our EditDistance function

This one is for comparing strings, by character



```
Func<Char, Char, double> func1 = (s, t) => 2.0;
```

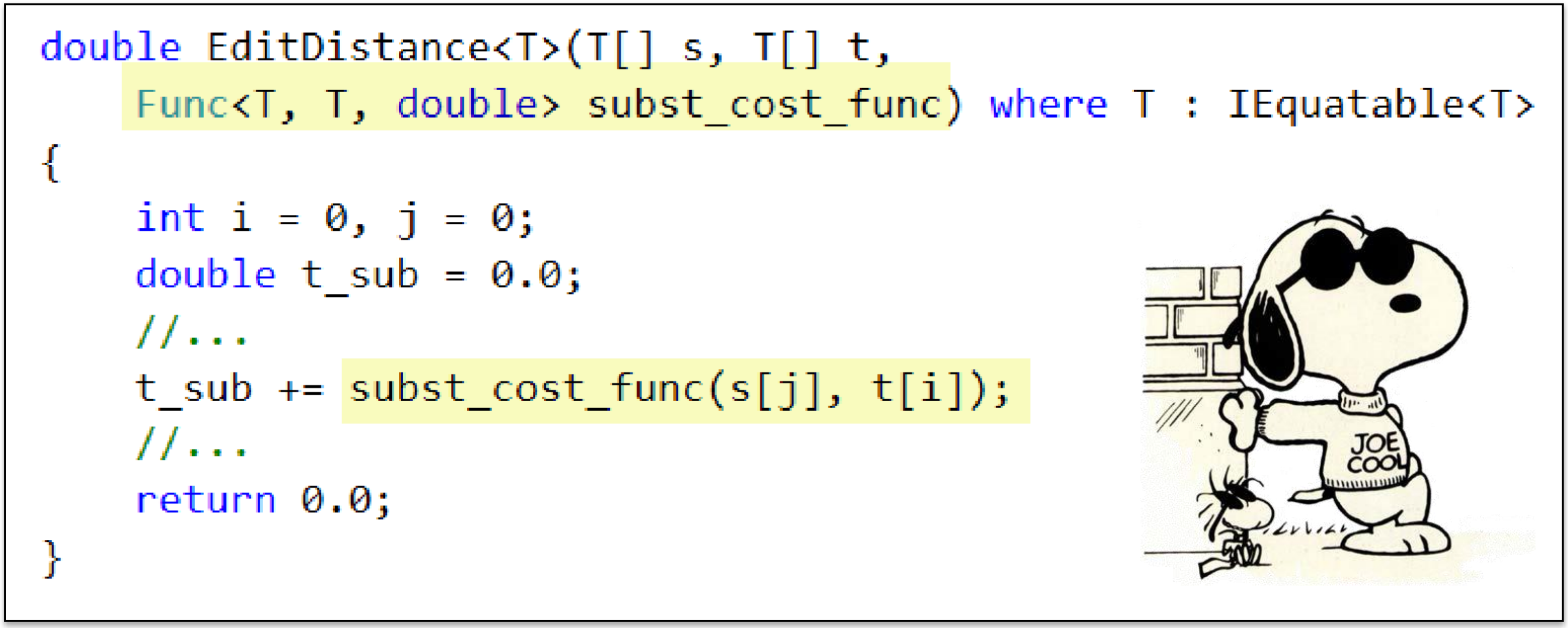
This one is for comparing entire texts, by line



```
Func<String, String, double> func2 = (s, t) =>
    {
        return 0.5 + EditDistance(s.ToCharArray(),
                                  t.ToCharArray());
    };
```

Putting it all together

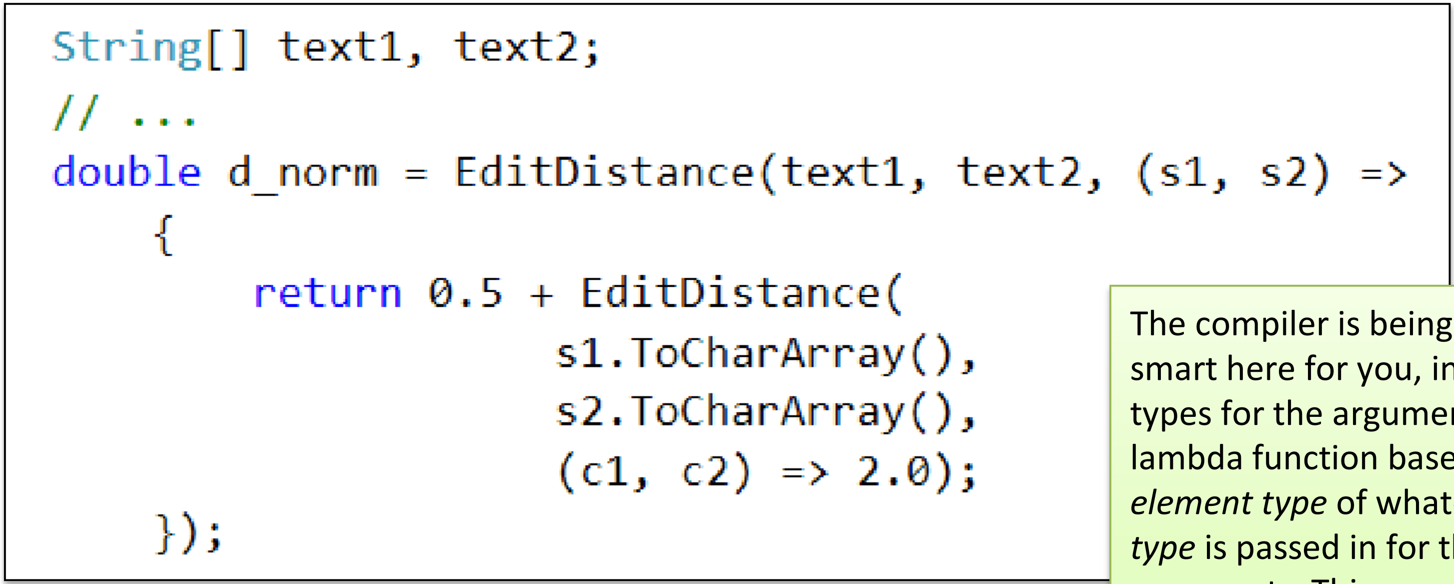
Now you’re ready to add another parameter to the EditDistance function: the substitution cost function—a lambda function—that the caller will pass into the function, in order to customize its behavior



```
double EditDistance<T>(T[] s, T[] t,
    Func<T, T, double> subst_cost_func) where T : IEquatable<T>
{
    int i = 0, j = 0;
    double t_sub = 0.0;
    //...
    t_sub += subst_cost_func(s[j], t[i]);
    //...
    return 0.0;
}
```

The grand finale

Now it all pays off: here's how to nest the calls to your type-agile template function, passing in the two different lambda functions, and getting the final result!



```
String[] text1, text2;
// ...
double d_norm = EditDistance(text1, text2, (s1, s2) =>
    {
        return 0.5 + EditDistance(
                    s1.ToCharArray(),
                    s2.ToCharArray(),
                    (c1, c2) => 2.0);
    });
```

The compiler is being really smart here for you, inferring the types for the arguments of the lambda function based on the *element type* of whatever *array type* is passed in for the first arguments. This saves you from having to explicitly specify types when you use a template.

Upcoming

- Next time
  - Guest lecture with Francis Bond
- Evaluation
  - Precision
  - Recall