

# Writing Homer's Basket

or,

Decisions, Decisions, Decisions

# Writing Homer's Basket

Remember when we sent Homer to the grocery store? We devised a very general algorithm for him, but parts of it were still very vague.

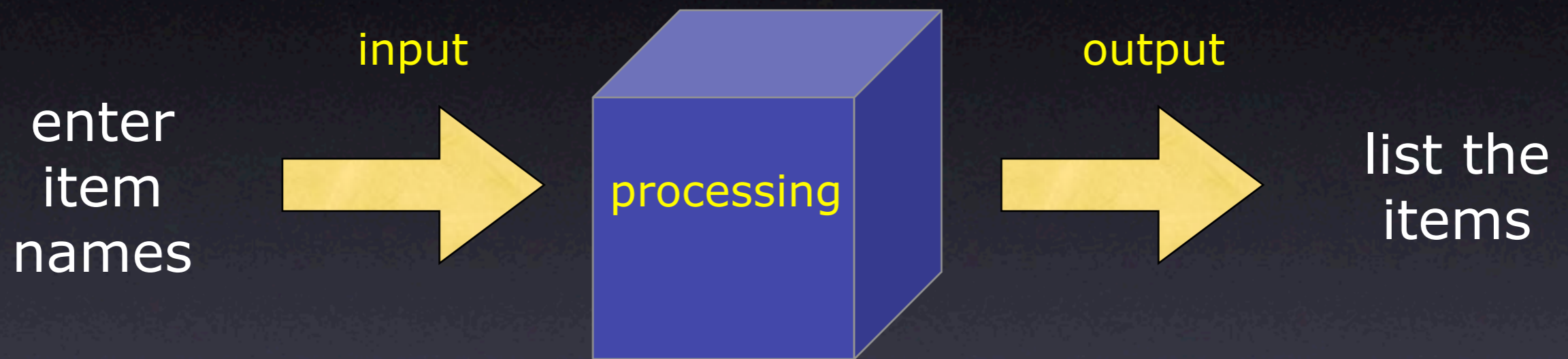
Let's write some Perl code that controls Homer's basket. (We'll skip the travel, checkout, etc.)

Here are the rules:

- The basket can contain no more than five items.
- When the basket contains five items, we need to stop adding items and list the contents of the basket.

# Input / Processing / Output

So here' what I'm thinking:



put items into some kind of variable for BASKET, count the items, make sure there are no more than 5. If not, get more items. Otherwise, output the list.

# Picking a data type

Let's think about that basket for just a moment. My choices there would be to use a scalar, an array or a hash.

- Since I'll have items (plural), and a scalar is singular, that probably won't work.
- A hash is plural, but I need to define both keys and values. I won't need the keys, so that's just extra work to give me a headache.
- An array has automatic indices, so that saves me work. Plus, they're numbered, which might be convenient because my basket holds only five items.

Let's go with an array: `@basket`.

# Counting

I also need a variable to store the number of items in the basket. It's a singular number, so a scalar will work. Maybe I'll just call it `$count`.

# Toward an Algorithm

Here's my first attempt at describing what I need to do:

1. Get an item from the user.
2. Put it in the basket.
3. Count the items in the basket.
4. If the count equals 5, print the list of items.
5. If the count is less than five, go to Step 1.

I can probably smooth this out a bit by swapping #4 and #5 (if the count is not less than five, logic would suggest that the count is five), but I'll do that later. #2 looks surprisingly tricky; let's think about that step for a moment.

# Put the Biscuit in the Basket

I know I can assign a scalar to an array element like this:

```
$basket[number] = $item;
```

But what if there are already 3 items in the basket? Then my assignment needs to look like this:

```
$basket[4] = $item;
```

But if I haven't yet counted the items, I have no idea which number to put between square brackets. So I need to count the items before I can put a new one in the basket.

Let's revise the algorithm.

# Toward an Algorithm

Here's my second attempt at an algorithm:

1. Count the items in the basket.
2. If the count equals 5, go to Step 6.
3. Get an item from the user.
4. Put it in the basket: `$basket[$count + 1]`
5. Go to Step 1.
6. Print the list of items in the basket.

I like this version. I'm a bit surprised that I should be counting items in Step 1, but doing so helps me to define Step 4. And the other logic problems have now disappeared all by themselves too.

# A Closer Look at "Go to"

Let's look more closely at a few selected steps:

2. If count is 5, go to ...

5. Go to Step ...

Perl doesn't have an explicit "Go To Step xxx" command, so we'll have to modify things a bit here to accommodate Perl. We've already seen a **foreach** loop. One option might be to use a variation on that strategy: a **while** loop.

```
while (a certain condition is true) {  
    # do stuff  
}
```

# A Closer Look at "Go to"

We want to keep adding items while there are fewer than five items in our basket. If there are already five items in the basket, we have to stop because we're not allowed to add any more. So we want to keep looping while `$count < 5`:

```
while ($count < 5) {  
    # add items to basket  
}
```

So the beginning and end of the **while** loop will satisfy Steps #2 and #5. Let's revise the algorithm again.

# Toward an Algorithm

Here's my third attempt at an algorithm:

1. Count the items in the basket.
2. while (`$count < 5`) do
3. Get an item from the user.
4. Put it in the basket: `$basket[$count + 1]`
5. end "while" loop (goes back to Step 2)
6. Print the list of items in the basket.

Now the algorithm is getting more refined, and it's even starting to look a little Perl-ish. But I see a problem: I'm not getting back to Step 1 so that I can count the basket again. We need another revision.

# Toward an Algorithm

There are a lot of ways to fix that problem. Here's one that's not very clever but is very easy. Version 4:

1. Count the items in the basket.
2. while (\$count < 5) do
3. Get an item from the user.
4. Put it in the basket: \$basket[\$count + 1]
5. Count the items in the basket.
6. end "while" loop (goes back to Step 2)
7. Print the list of items in the basket.

And I can think of one thing we forgot...

# Algorithm version 5

1. Establish @basket and initialize it as empty.
2. Establish \$count and initialize it to zero.
3. while (\$count < 5) do
  4. Get an \$item from the user.
  5. Put \$item in the basket: \$basket[\$count + 1]
  6. Count the items in the basket.
7. end "while" loop (goes back to Step 3)
8. Print the list of items in the basket.

Now I'd work through it myself with pencil and paper.

A Brief Interlude  
to Run Some Perl

# The Process

Programming is a bit like juggling: you'll have a lot of balls in the air and when you're doing it, it's hard to remember how and where you started. Nonetheless, you'll have to juggle the following balls:

- Define your **Input** and your **Output**.
- Devise and revise your **algorithm** to move Homer successfully from Input to Output.
- Think about the **data types** you want to use. A bad choice here can make your job harder. A good choice here can solve problems for you.
- Get the basic program working before you add features. Take small steps and **solve one problem at a time**. Don't do everything at once.

# What to Hand In

1. Your **algorithm**. Work through it with pencil and paper to make sure it's accurate before you translate it into Perl. It's useful to number the Steps of your algorithm.
2. Your **program**. You can send it to me as an email attachment. You'll get graded on correctness (does it give the right answer?), style (I should be able to read your program easily), comments (including labeling the steps of your algorithm) and even creativity.
3. A **post-mortem analysis**. Describe for me what you felt was successful and unsuccessful about this Perl project. You can combine #1 and #3 into one document if you wish.