CSY2028
Web Programming
Topic 11

Tom Butler
thomas.butler@northampton.ac.uk

Topic 11

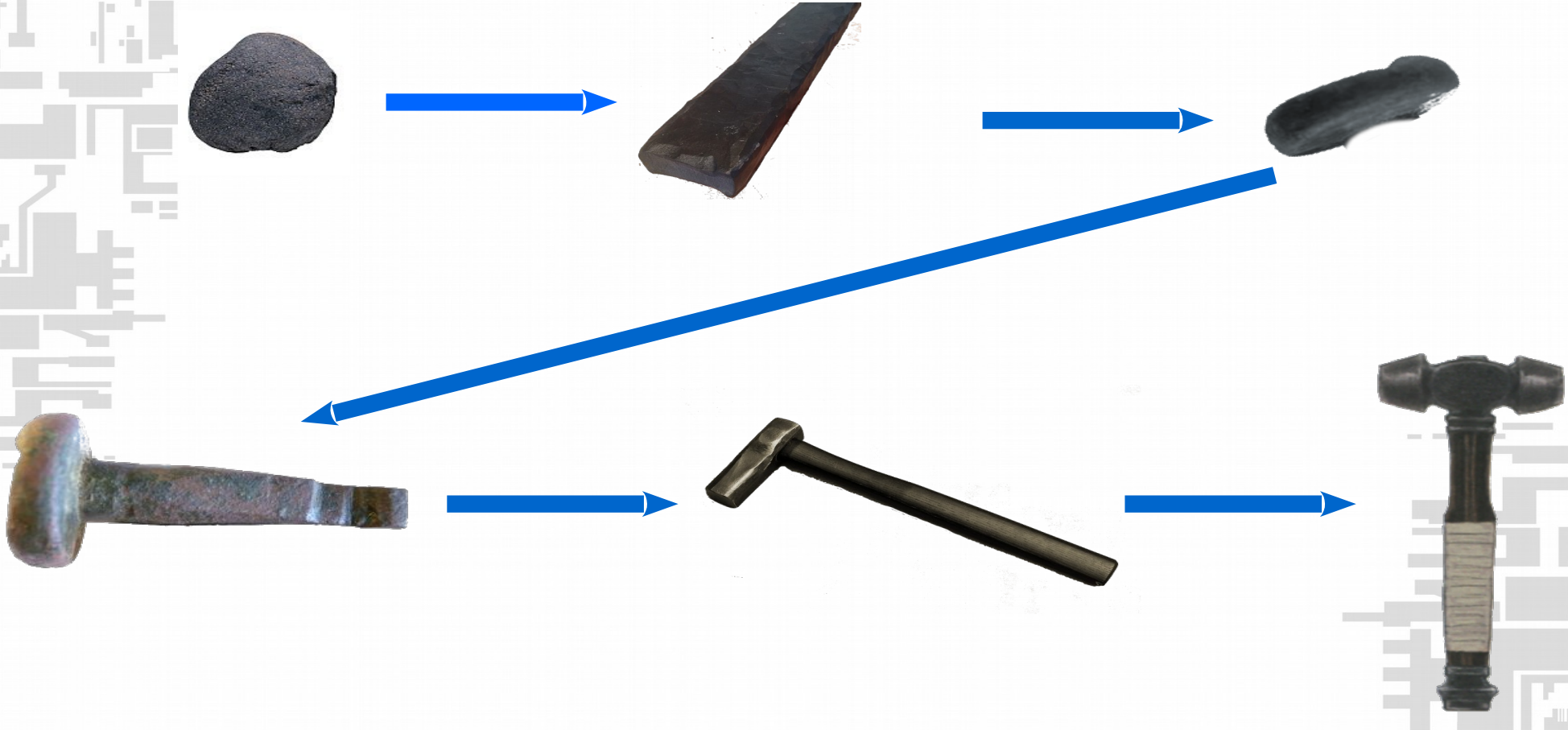
- Intro to term 2
- How to think about programming concepts
- Writing reusable functions
- Abstracting SQL queries

Term 2

- Term 2
- Last term was focussed on how to write PHP code and getting some working code together
- You have the basic tools you need to create a website
- Term 2's main focus will be on structuring the code
- As you were building your assignment you might have found yourself needing to repeat code
- Or needing to build different pages that perform similar tasks
- Term 2 is focussed on breaking these problems up into small, reusable chunks and different methods of doing so
- How to think about programming

“Blacksmiths are unique in that they are able to make their own tools”

- Daniel Dennet (Intuition Pumps and Other Tools for Thinking)



Improving tools

Better tools allows for

- Faster completion of products
- Better quality final products
- A wider variety of products to be produced
- Less skilled workers to make the products
- *Other , specialised tools to be produced*

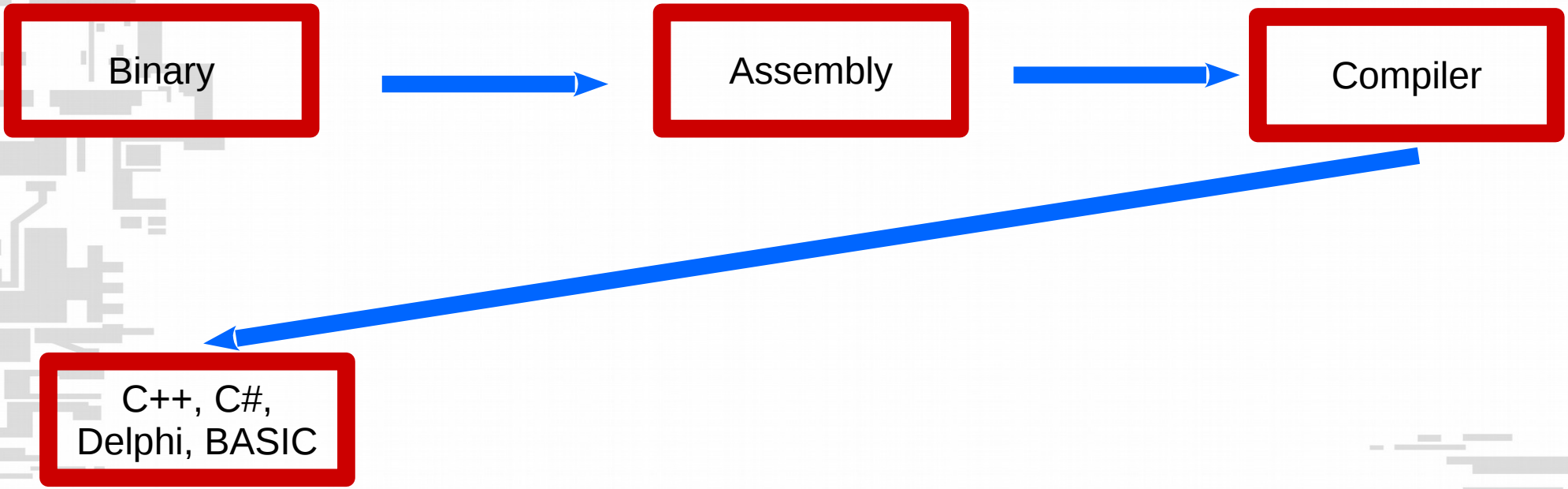
(Kayne & Son Blacksmiths Depot n.d.)

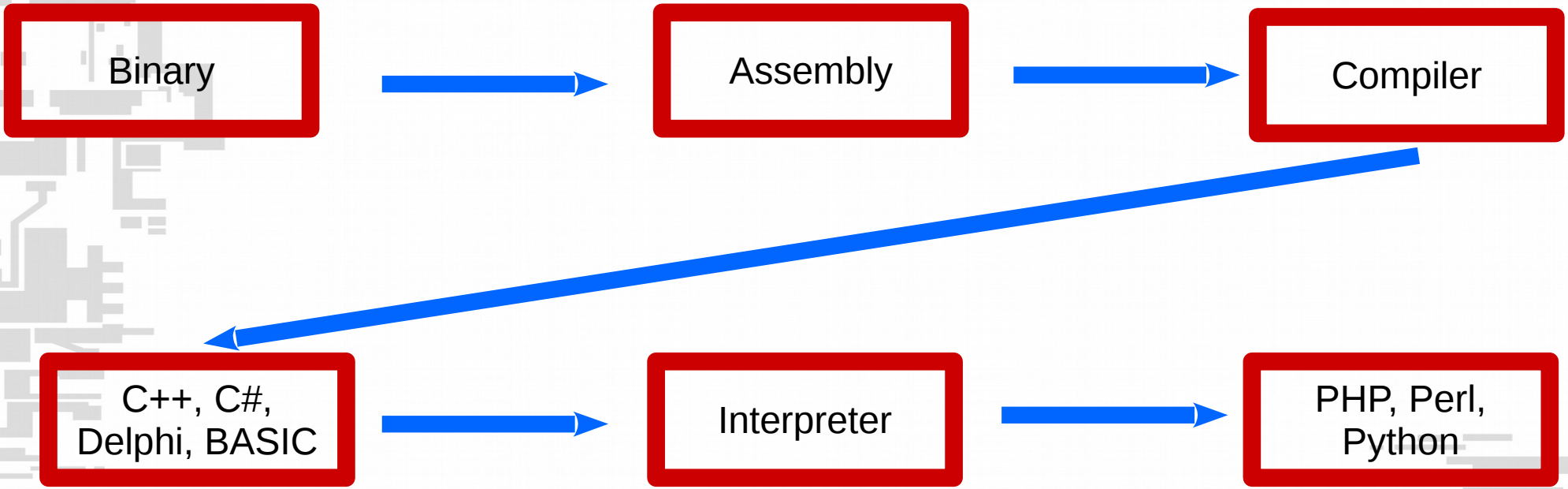
Reusability

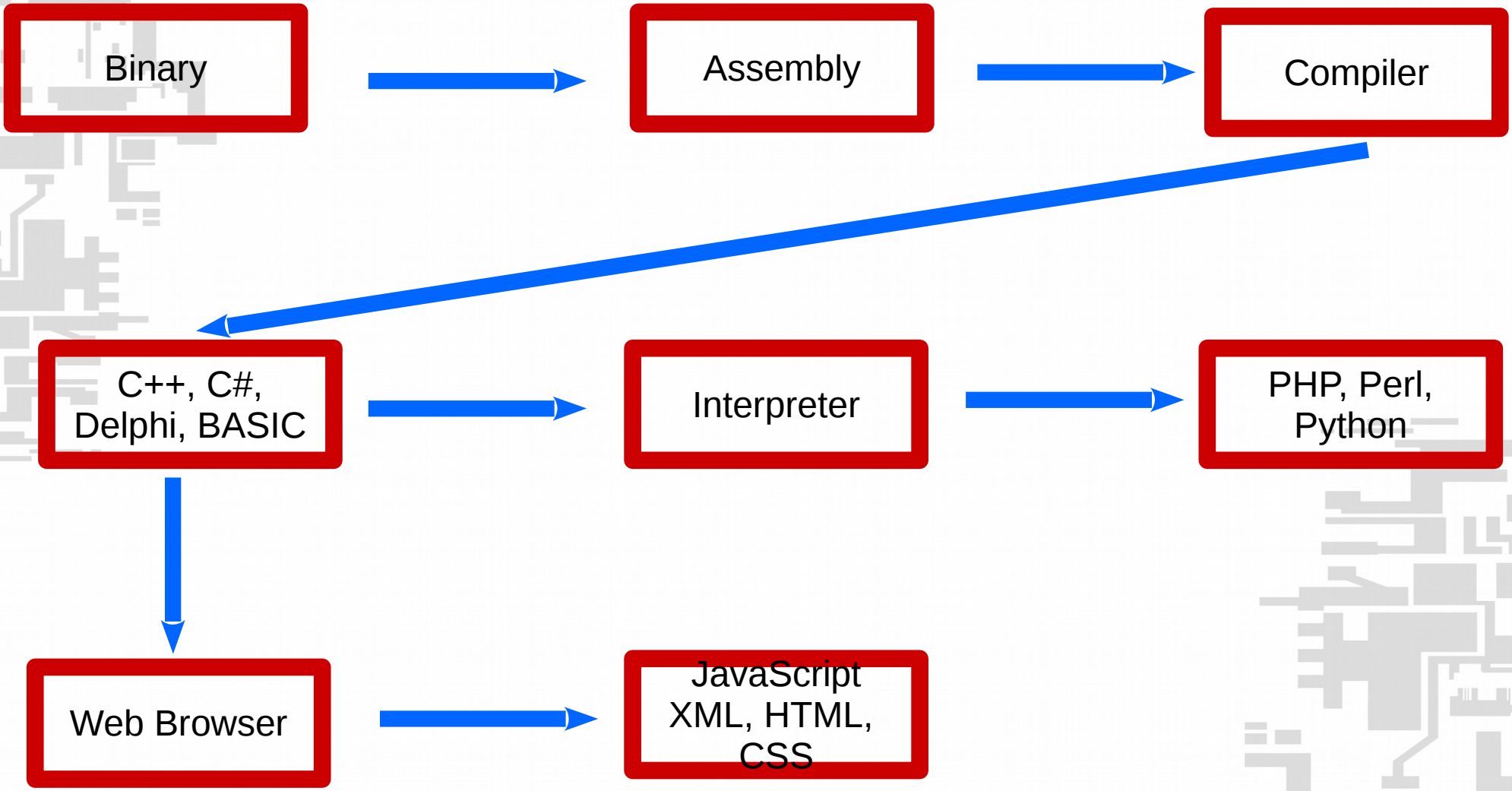
- Tools can be reused
  - Once you have a good hammer you can use it to make thousands of products
  - It takes time upfront to make the tool but saves time in the long run

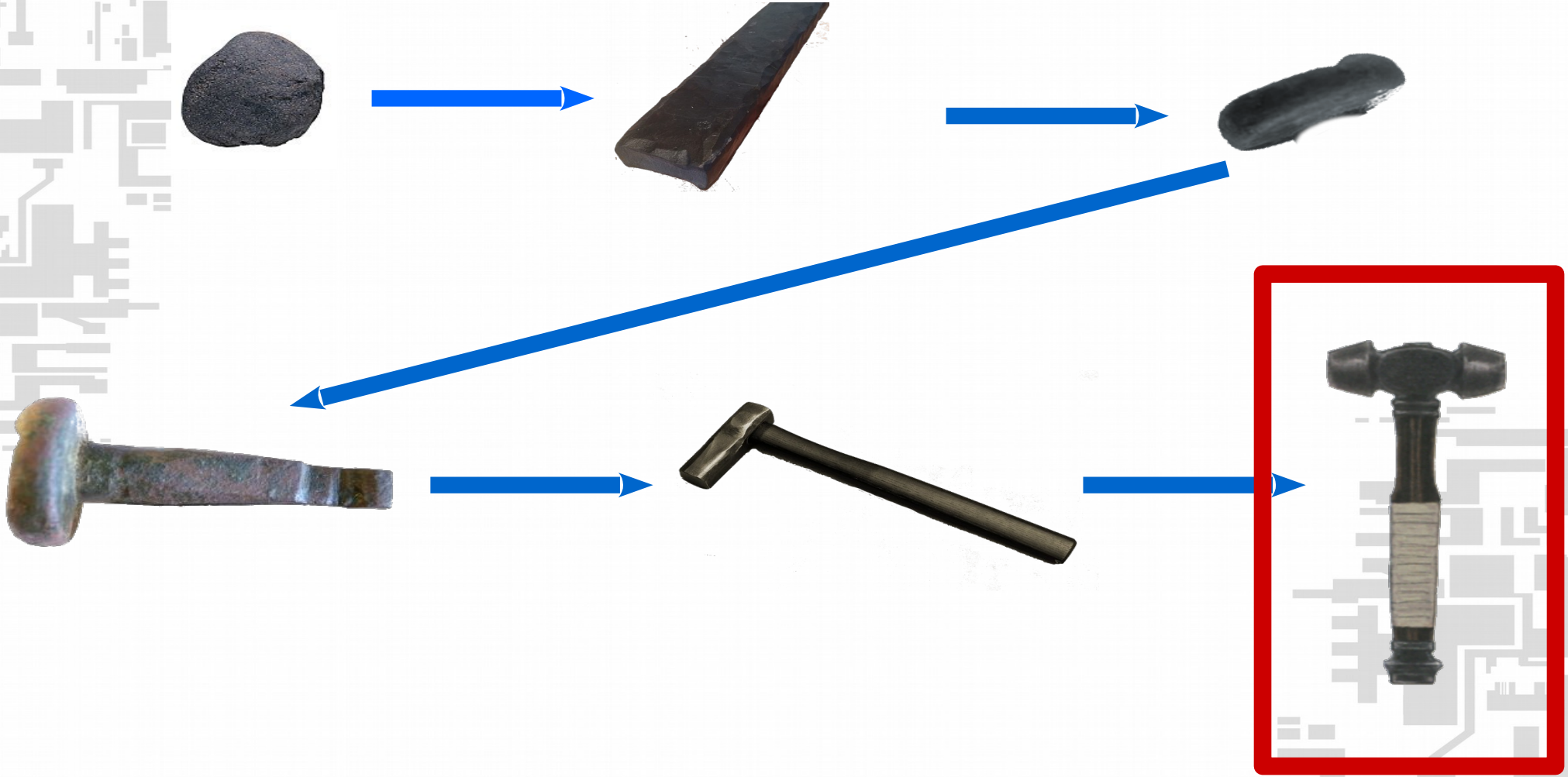
Blacksmiths are <u>almost</u> unique in that they are able to make their own tools


Binary
Assembly


Binary
Assembly
Compiler
C++, C#,
Delphi, BASIC


Binary
Assembly
Compiler
C++, C#,
Delphi, BASIC
Interpreter
PHP, Perl,
Python


Binary
Assembly
Compiler
C++, C#,
Delphi, BASIC
Interpreter
PHP, Perl,
Python
Web Browser
JavaScript
XML, HTML,
CSS



Tools

- Every time you write a *function* you are *creating a tool*.
- That tool then lets you build the program
- The tool could also help you build other programs
  - *Unless your tool is too specific and can't be used for anything else! What makes a good tool?

Tools

- As you become a better programmer you find yourself using even more advanced tools
  - IDEs
  - Unit Testing suites
  - Databases
  - Libraries for generating HTML
  - etc
- You can even write your own implementations of these or create any tool you need to help you with your job

What makes a good tool?

- Each time you write a function or a class you should think about how it is being used
- By making it *generic* and usable in more than one circumstance you are saving yourself work later on
- If you have to solve very similar problems over and over you should try to write the code once in a way that it can be used repeatedly
- **Most websites involve solving very similar problems!**

“A worker is only as good as his tools”

Tools

- You can make your own tools using the tools provided for you
  - Functions and variables are tools provided by the language
  - Arguments and return values are also provided by the language
- You can use these to create your own tools
- And then you can use those tools as many times as you like

Tools

- Any repeated task can be moved into a function
- When coding your assignment you probably had to find a user by their ID several times using code similar to this:

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE id = :id');
$criteria = [
        'id' => $id
];
$stmt->execute($criteria);

$user =  $stmt->fetch();
```

Functions

- By moving the code into a **function** it becomes reusable:

```
function findPersonById($pdo, $id) {
        $stmt = $pdo->prepare('SELECT * FROM person WHERE id = :id');
        $criteria = [
                'id' => $id
        ];
        $stmt->execute($criteria);

        return $stmt->fetch();
}

$person = findPersonById($pdo, 123);

echo $person['firstname'];
echo $person['surname'];
```

- The `findByPerson` function is a useful tool, you can use it anywhere you need to find a user by their ID:

```
$person = findPerson($pdo, 123);

$person = findPerson($pdo, $_GET['userid'];);
```

INSERT Queries

- To insert data into a database, you used the code

```
$stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                                           VALUES (:email, :firstname, :surname, :birthday)
');

$criteria = [
        'firstname' => $_POST['firstname'],
        'surname' => $_POST['surname'],
        'email' => $_POST['email'],
        'birthday' => $_POST['birthday']
];

$stmt->execute($criteria);
```

- Each time you want to write to the person table you will need to repeat this code

INSERT function

- Instead, it's possible to write a function to do this for you:

```
function insertPerson($pdo, $criteira) {
        $stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                                        VALUES (:email, :firstname, :surname, :birthday)
        ');


        $stmt->execute($criteria);
}
```

- Which can be used to insert as many records as you like by only providing the $criteria variable and $pdo object

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

$person2 = [
        'firstname' => 'Sue',
        'surname' => 'Jones',
        'email' => 'sue@example.org',
        'birthday' => '1992-02-21'
];
insertPerson($pdo, $person1);
insertPerson($pdo, $person2);
```

Delete function

- This can also be applied to a DELETE query

```
function deletePerson($pdo, $id) {
        $stmt = $pdo->prepare('DELETE FROM person WHERE id = :id');
        $criteria = [
                'id' => $id
        ];
        $stmt->execute($criteria);
}

deletePerson($person, 1);
deletePerson($person, 4);
```

- deletePerson($pdo, $id); can be used to quickly delete a single record from the person table
- Anywhere you need to delete a person from the database you can call deletePerson() rather than typing out the query

Tools

- Now, we have a set of reusable functions:
  - findPersonById() - returns a record from the `person` table by the ID of the record
  - insertPerson() - Adds a record to the database
  - deletePerson() - deletes a person from the database

Exercise 1

- 1) Write similar functions for one of your tables. Either use one of the tables from your assignment or create one (e.g. the person table from the examples)
- 2) Test the `findPersonById` function by using it to select multiple records from the table.
- 3) Test the `deletePerson` and `insertPerson` functions

Better select function

- The `findPerson` function only allows selecting records by ID
- It's possible to extend the function with an extra argument to allow searching for a different field

```
function findPerson($pdo, $field, $value) {
        $stmt = $pdo->prepare('SELECT * FROM person WHERE ' . $field . ' = :value');
        $criteria = [
                'value' => $value
        ];
        $stmt->execute($criteria);

        return $stmt->fetch();
}

//Find user where id is equal to 123
$person = findPerson($pdo, 'id', 123);


//Find user where email is equal to 'john@example.org'
$person = findPerson($pdo, 'email', 'john@example.org');
```

Better select function

- Now, we can replace and SELECT query for the person table:

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE firstname = :firstname');
$criteria = [
        'value' => $value
];
$stmt->execute($criteria);

$person = $stmt->fetch();
```

```
$person = findPerson($pdo, 'firstname', $value);
```

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE id = :id');
$criteria = [
        'value' => $value
];
$stmt->execute($criteria);

$person = $stmt->fetch();
```

```
$person = findPerson($pdo, 'id', $value);
```

Update function

- It's possible to do the same thing with an update function:

```
function updatePerson($pdo, $criteira, $updateField, $updateValue) {
        $stmt = $pdo->prepare('UPDATE person
                SET email = :email,
                firstname = :firstname,
                surname = :surname,
                birthday = :birthday
                WHERE ' . $updateField . ' = :updateValue';

        $criteria['updateValue'] = $updateValue;

        $stmt->execute();
}


$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];


//Update the record with the ID of `123` with the information in $person
updatePerson($pdo, $person1, 'id', 123);
```

Reusable functions

- We now have quick and easy to use functions for selecting, updating, inserting and deleting records from the person table:

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

//Update the record with the ID of `123` with the information in $person
updatePerson($pdo, $person1, 'id', 123);

//Find user where id is equal to 123
$person = findPerson($pdo, 'id', 123);


//Find user where email is equal to 'john@example.org'
$person = findPerson($pdo, 'email', 'john@example.org');


insertPerson($pdo, $person1);

//Delete the person with the ID 123
deletePerson(123);
```

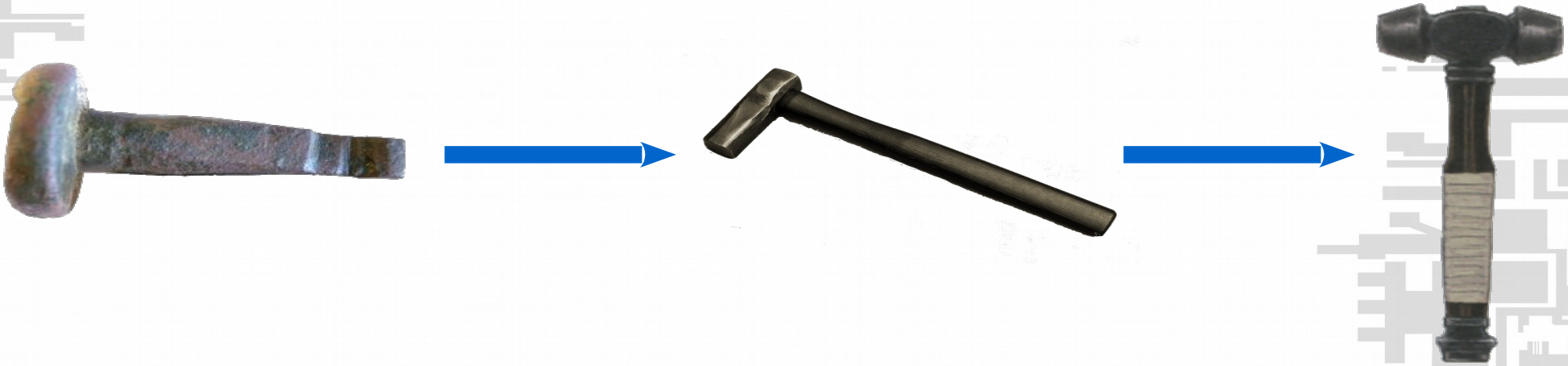
Reusable functions

- It's a lot quicker and easier to insert/update/delete/select from the person table as each can be done with a single line of code

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

//Update the record with the ID of `123` with the information in $person
updatePerson($pdo, $person1, 'id', 123);

//Find user where id is equal to 123
$person = findPerson($pdo, 'id', 123);


//Find user where email is equal to 'john@example.org'
$person = findPerson($pdo, 'email', 'john@example.org');


insertPerson($pdo, $person1);

//Delete the person with the ID 123
deletePerson(123);
```

Reusable functions

- These functions are a lot more reusable than code without them however they can be refined further



Improving the functions

- Instead of writing functions to interact with the *person* table it's possible to rewrite the function to work with *any* table
- This can be done by adding an extra argument to each function call

Find function

```
function findPerson($pdo, $field, $value) {
        $stmt = $pdo->prepare('SELECT * FROM person WHERE ' . $field . ' = :value');
        $criteria = [
                'value' => $value
        ];
        $stmt->execute($criteria);

        return $stmt->fetch();
}


function find($pdo, $table, $field, $value) {
        $stmt = $pdo->prepare('SELECT * FROM ' . $table . ' WHERE ' . $field . ' = :value');
        $criteria = [
                'value' => $value
        ];
        $stmt->execute($criteria);

        return $stmt->fetch();
}
```

Enhanced SELECT function

```
function find($pdo, $table, $field, $value) {
        $stmt = $pdo->prepare('SELECT * FROM ' . $table . ' WHERE ' . $field . ' = :value');
        $criteria = [
                'value' => $value
        ];
        $stmt->execute($criteria);

        return $stmt->fetch();
}

//Find a record from the person table where the id field is 123
$person = find($pdo, 'person', 'id', 123);
echo $person['surname'];

//Find a record from the job table where the id field is 22
$job = find($pdo, 'job', 'id', 22);
echo $job['title'];

//Find a record from the job table where the id field is 22
$applicant = find($pdo, 'applicant', 'email', 'john@example.org');
echo $applicant['name'];
```

Enhanced SELECT function

```
function find($pdo, $table, $field, $value) {
        $stmt = $pdo->prepare('SELECT * FROM ' . $table . ' WHERE ' . $field . ' = :value');
        $criteria = [
                'value' => $value
        ];
        $stmt->execute($criteria);

        return $stmt->fetch();
}
```

- This allows you to search:
  - Any table in the database
  - Using any single field
  - And any value
  - And retrieve the first result

Enhanced SELECT function

```
function find($pdo, $table, $field, $value) {
        $stmt = $pdo->prepare('SELECT * FROM ' . $table . ' WHERE ' . $field . ' = :value');
        $criteria = [
                'value' => $value
        ];
        $stmt->execute($criteria);

        return $stmt->fetch();
}
```

- Generally, however we won't want just the first result so the entire $stmt should be returned

```
function find($pdo, $table, $field, $value) {
        $stmt = $pdo->prepare('SELECT * FROM ' . $table . ' WHERE ' . $field . ' = :value');
        $criteria = [
                'value' => $value
        ];
        $stmt->execute($criteria);

        return $stmt;
}
```

- We can also add a function for finding all the records in the table without a WHERE clause:

```
function findAll($pdo, $table) {
        $stmt = $pdo->prepare('SELECT * FROM ' . $table );

        $stmt->execute();

        return $stmt;
}

//Find a record from the person table where the id field is 123
$records = findAll($pdo, 'person');
foreach ($records as $person) {
///…
}
```

Inserts and Updates

- The same can be done with INSERT and UPDATE queries
- But it will take a bit more work
- INSERT and UPDATE queries require knowledge of the field names being written to
- e.g. The insertPerson function

```php
function insertPerson($pdo, $criteira) {
        $stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                                                VALUES (:email, :firstname, :surname, :birthday)
        ');


        $stmt->execute($criteria);
}
```

```php
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

$person2 = [
        'firstname' => 'Sue',
        'surname' => 'Jones',
        'email' => 'sue@example.org',
        'birthday' => '1992-02-21'
];

insertPerson($pdo, $person1);
insertPerson($pdo, $person2);
```

Inserts and Updates

- Is it possible write a generic insert function that can be used to write to any table?
- Somehow you will need to know the field names in the table to create the query
- However, these are provided in the $criteria array

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];


insertPerson($pdo, $person1);
```

Inserts and Updates

- The keys of the $person1 array includes all the field names needed to perform the INSERT

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];


insertPerson($pdo, $person1);
```

Inserts and Updates

- The PHP function array_keys() returns an array of all the *keys* of a specified array

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

$keys = array_keys($person1);

var_dump($keys);
```

```
Output:
array (size=4)
  0 => string 'firstname' (length=9)
  1 => string 'surname' (length=7)
  2 => string 'email' (length=5)
  3 => string 'birthday' (length=8)
```

Inserts and Updates

- The php implode() function takes an array and joins the contents with an optional separator

```php
$array = ['One', 'Two', 'Three'];

$string = implode(', ', $array);

echo $string;
```

```
Output:
One, Two, Three
```

Note the first argument of the implode() function is ', '
This will put a comma and a space between each element when it is joined

Inserts and updates

- By combing array_keys and implode() it's possible to generate the first part of the INSERT query based on the $person1 array

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

$keys = array_keys($person1);

$implodedString = implode(', ', $keys);

echo 'INSERT INTO person (' . $implodedString  . ')';
```

```
Output:
INSERT INTO person VALUES (firstname, surname, email, birthday)
```

Inserts and updates

- The second part of the query is more complicated, however implode can be used again:

```
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

$keys = array_keys($person1);

$implodedString = implode(', :', $keys);

echo ':' . $implodedString ;
```

```
Output:
:firstname, :surname, :email, :birthday
```

By imploding using
`, :`
A colon precedes almost every element

Note: the string is prefixed
With a :
Implode only adds

Inserts and updates

- This can be put together to generate the entire INSERT query

```php
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

$keys = array_keys($person1);

$values = implode(', ', $keys);
$valuesWithColon = implode(', :', $keys);

$query = 'INSERT INTO person (' . $values . ') VALUES (:' . $valuesWithColon . ')';

echo $query;
```

```
Output:
INSERT INTO person (firstname, surname, email, birthday) VALUES (:firstname, :surname, :email, :birthday)
```

Inserts and updates

- By using a variable in place of the table name it's possible to write an insert function that will alow inserting into any database table

```
function insert($pdo, $table, $record) {
        $keys = array_keys($record);

        $values = implode(', ', $keys);
        $valuesWithColon = implode(', :', $keys);

        $query = 'INSERT INTO ' . $table . ' (' . $values . ') VALUES (:' . $valuesWithColon . ')';

        $stmt = $pdo->prepare($query);

        $stmt->execute($record);
}
```

Inserts and updates

```php
function insert($pdo, $table, $record) {
        $keys = array_keys($record);

        $values = implode(', ', $keys);
        $valuesWithColon = implode(', :', $keys);

        $query = 'INSERT INTO ' . $table . ' (' . $values . ') VALUES (:' . $valuesWithColon . ')';

        $stmt = $pdo->prepare($query);

        $stmt->execute($record);
}
```

```php
$person1 = [
        'firstname' => 'John',
        'surname' => 'Smith',
        'email' => 'john@example.org',
        'birthday' => '1989-12-02'
];

insert($pdo, 'person', $person1);

$job1 = [
        'title' => 'Assistant manager',
        'job_ref' => '1333',
        'description' => 'Assistant to te manager',
        'salary' => '25,000'
];

insert($pdo, 'job', $job1);
```

Reusability

- This function allows for you to insert a record into any database table
- By writing similar functions for update/delete/select it's possible to very quickly build an application without ever writing an SQL query!
- Most web sites in PHP will need this functionality, if you put these functions in their own file you can easily transfer them between websites
- Once you have these tools built you can use them as many times as you like

Resusability

- Most websites require solving very similar (or identical!) problems:
  - Interacting with databases
  - Handling user log ins/accounts
  - Administration areas
  - Updating/adding content via the website
  - Generating HTML
    - Wrapping content in a standard header/footer
    - Displaying forms
  - Processing HTML form submissions
    - Validating HTML form submissions
    - Inserting data from forms into a database

Programmers are lazy

- Because these tasks are incredibly common it's worth writing the code once with a view to using it on more than one website
- It takes a little thought up-front but allows you to use the same code on more than one website
- Once you've written the code you can use it over and over

Writing reusable code

- To write reusable code you have to be careful not to write the code in a way that makes it specific to a single task or website
- E.g. the insertPerson() function from earlier

```
function insertPerson($pdo, $criteira) {
        $stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                                                VALUES (:email, :firstname, :surname, :birthday)
        ');


        $stmt->execute($criteria);
}
```

- This can only be used on websites that have a table called `person` with these exact fields

Writing reusable code

- However, the generic insert() function can be used with any table on any website:

```
function insert($pdo, $table, $record) {
        $keys = array_keys($record);

        $values = implode(', ', $keys);
        $valuesWithColon = implode(', :', $keys);

        $query = 'INSERT INTO ' . $table . ' (' . $values . ') VALUES (:' . $valuesWithColon . ')';

        $stmt = $pdo->prepare($query);

        $stmt->execute($record);
}
```

- This function is more complicated to write, but it will save time going forward

Next week...

- I'll show you how to refine these functions even further to make a very simple way of quickly editing and updating records

Exercise 2

- 1) Download exercise2.zip and amend the code to use the generic INSERT and SELECT functions (provided in functions.php) in place of repeating the SQL queries
  - Hint: I recommend setting up a new server in a new directory. Create a new folder for this project and extract the `website` folder into it before between running `vagrant init` and `vagrant up`. If you do it this way the sample database will be created for you!
- 2) Can you create a generic UPDATE and DELETE function? Solution in next week's lecture.