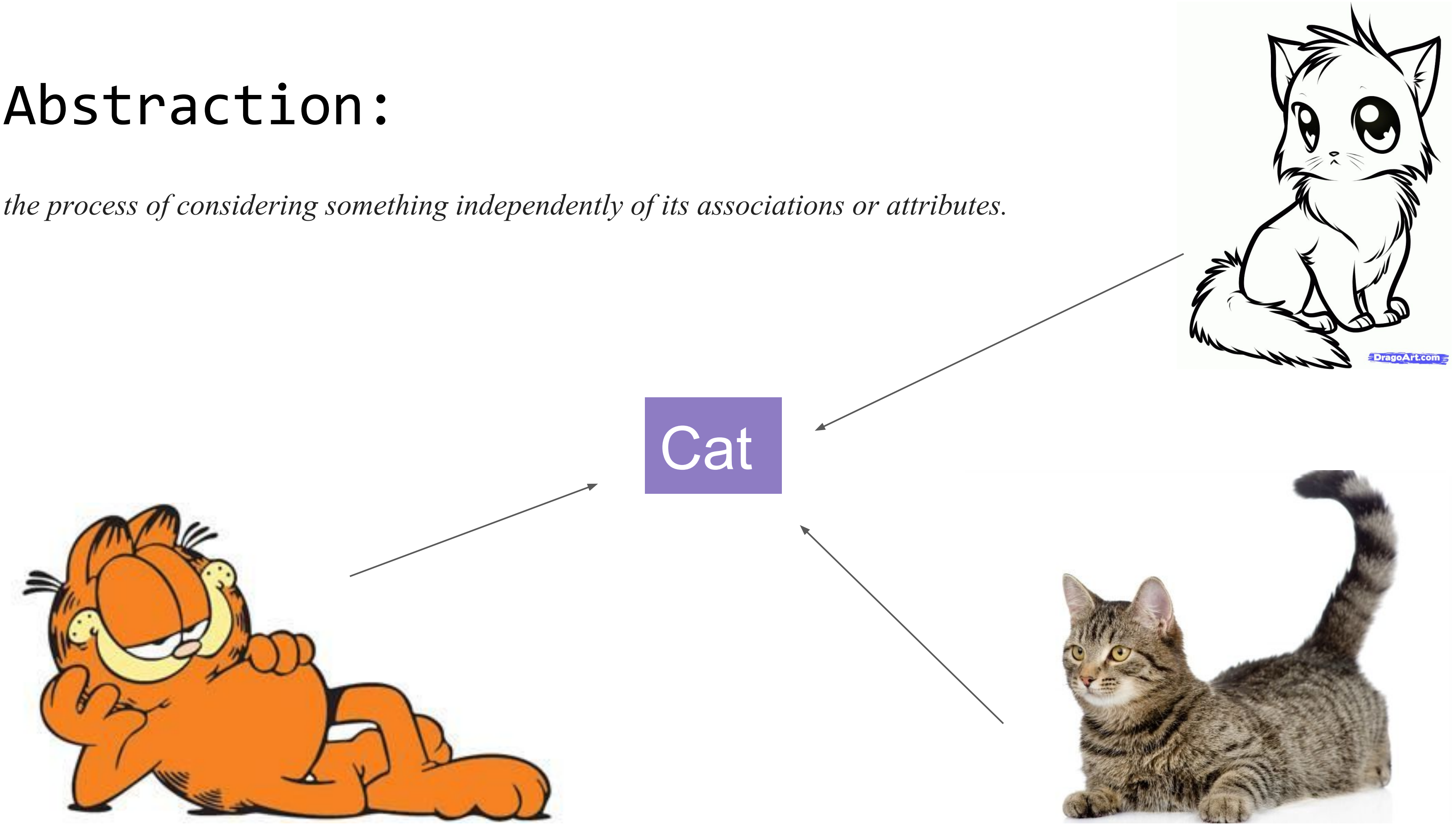
# Abstractions

## Abstraction:

*the process of considering something independently of its associations or attributes.*



## In OOP

In OOP, abstraction is the process of identifying essential, common concepts (usually as objects) to allow further complexity to be layered over the top. The purer the abstraction, the less coupled it is to any immediate context. Abstractions can be made out of other abstractions:

- Garfield (instance)
- Persian
- Cat
- Mammal
- Vertebrate...

It goes hand-in-hand with encapsulation, the process of hiding an object's data, state or implementation to allow a simpler interface for a user. We don't need to care about the inner workings of a cat to talk about one.

## In code

Abstractions don’t always have to be objects, or have anything to do with OOP. They can be:

- Objects
- Functions, especially curried functions
- Modules
- Libraries (e.g Lodash) or frameworks (e.g. Express)

```
import request from 'superagent'

const submitForm = (form) => {
  if (form.values === null) return new Error('blah');

  if (form.values.firstName === null) return new Error('name');

  return request.put('some-url', JSON.stringify(form))
    .use(superagentPromisePlugin)
    .withCredentials()
    .timeout(9000)
    .set({
      Accept: 'application/json',
      'Content-Type': 'application/json',
    })
    .then(response => store.write(response))
    .catch(error => {
      if (error.status === 400 && isValidationErr) {
        return { validationErrors: error.response.body.errors };
      }

      if (error.status === 400 && isTypeChangeErr) {
        return error.response.body;
      }

      if (error.status === 401) {
        return { userUnauthorised: true };
      }
      return { serverError: true };
    });
}
```

## Few abstractions… What if we had more than one PUT request?

```
import request from 'superagent'

const putForm = (endpoint, body) =>
  request.put(endpoint, 'PUT', JSON.stringify(body))
    .use(superagentPromisePlugin)
    .withCredentials()
    .timeout(9000)
    .set({
      Accept: 'application/json',
      'Content-Type': 'application/json',
    })
    .then(() => ({}))
    .catch((error) => {
      const isTypeChangeErr = _get(error, 'response.body.typeChangeError');

      if (error.status === 400 && isValidationErr) {
        return { validationErrors: error.response.body.errors };
      }

      if (error.status === 400 && isTypeChangeErr) {
        return error.response.body;
      }
      return { serverError: true };
    });


const submitForm = (form) => {
  if (form.values === null) return new Error('blah');

  if (form.values.firstName === null) return new Error('name');

  return putForm('some-url', form);
}
```

What if we had more than just PUTs? Or needed to remove superagent?

```
import request from 'superagent'

const request = (endpoint, method, body, query) => {
  const methods = {
    PUT: superagent.put,
    GET: superagent.get,
  };

  const req = methods[method](endpoint)
    .use(superagentPromisePlugin)
    .withCredentials()
    .timeout(9000)
    .set({
      Accept: 'application/json',
      'Content-Type': 'application/json',
    })
    .query(query);

  if (body) {
    req.send(body);
  }

  return req;
};
```

Abstraction: a generic, reusable request function provides a layer over superagent.

## A http request in Node, using http

```
const https = require('https');

https.get('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY', (resp) => {
  let data = '';

  // A chunk of data has been received.
  resp.on('data', (chunk) => {
    data += chunk;
  });

  // The whole response has been received. Print out the result.
  resp.on('end', () => {
    console.log(JSON.parse(data).explanation);
  });

}).on("error", (err) => {
  console.log("Error: " + err.message);
});
```

We have to deal with data arriving in chunks, would need a separate module for HTTPS, would need to parse our own JSON, etc.

# A http request using Axios (with more abstraction)

```
const axios = require('axios');

axios.get('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY')
  .then(response => {
    console.log(response.data.url);
    console.log(response.data.explanation);
  })
  .catch(error => {
    console.log(error);
  });
```

Here we're parsing JSON by default, and automatically storing the response as one object once it's finished arriving. Promises also allow us to more elegantly handle async code.

Frameworks like Express similarly allow us to write APIs without having to deal with data chunking, Buffers or any of the underlying events of Node.

## Reading

https://css-tricks.com/importance-javascript-abstractions-working-remote-data/

https://netbasal.com/the-importance-of-abstraction-in-js-ea27e07e996

https://medium.com/javascript-scene/abstraction-composition-cb2849d5bdd6

https://www.twilio.com/blog/2017/08/http-requests-in-node-js.html