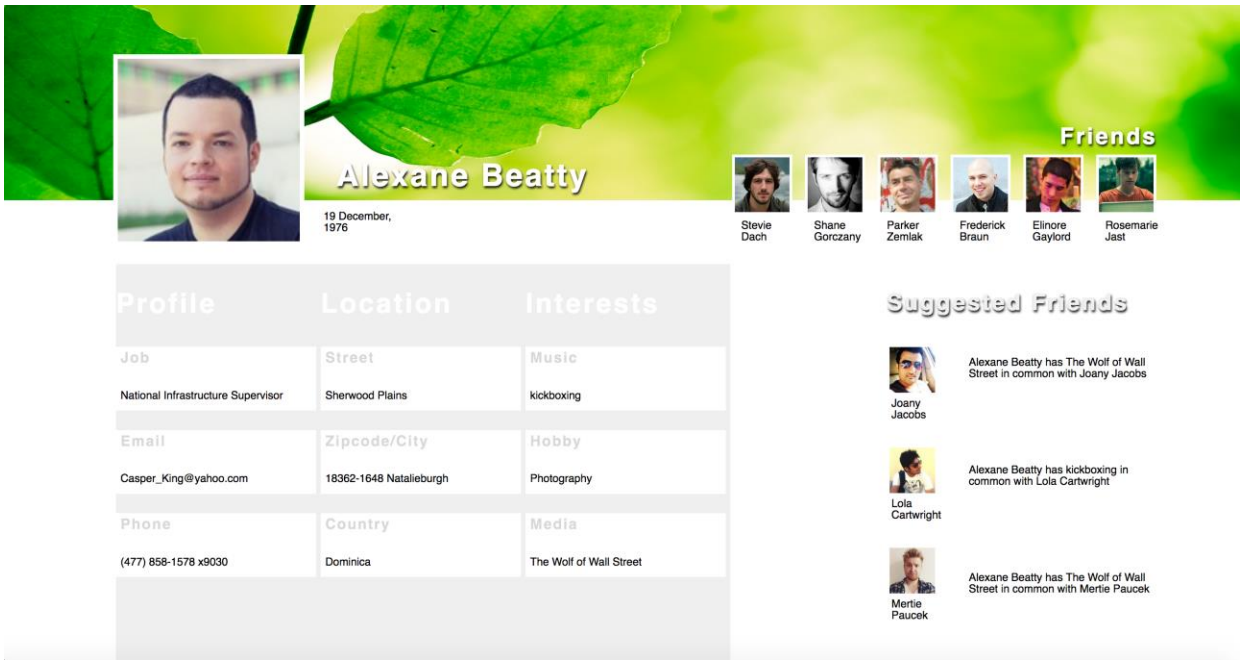
Problematic Profile Personification



A Rawgit URL for our program:

https://cdn.rawgit.com/Magnusaur/wizards_of_buzz.exe/6b138d5f/AP_Exam/index_exam.html

A video demonstration of our program:

https://www.youtube.com/watch?v=81elyEGZcbs&feature=youtu.be

Faker.js Github repository: https://github.com/marak/Faker.js/

Overview

Object-oriented programming serves as a widely used programming paradigm in software development. By extension, object-orientation in general has become a prevalent ontological approach to conceptualizing the world. The ambition of this project is to explore and critically reflect on the practice of object-orientation in programming and computational culture. In what follows, we will briefly introduce what exactly our project is, how it works and why we made it.

The what

Our project takes the form of a mock profile page on faux website. The layout of the website imitates that of various social media pages. The profile comprises various categories of personal information, including a name and a profile picture. In addition to the various information attached to it, the profile is also seemingly connected to other pseudo-profiles, labeled 'friends'. Another category of potential connections, 'suggested friends', is also presented on the page, based on common interests. Either type of friends can be clicked in order to access the individual profiles.

The how

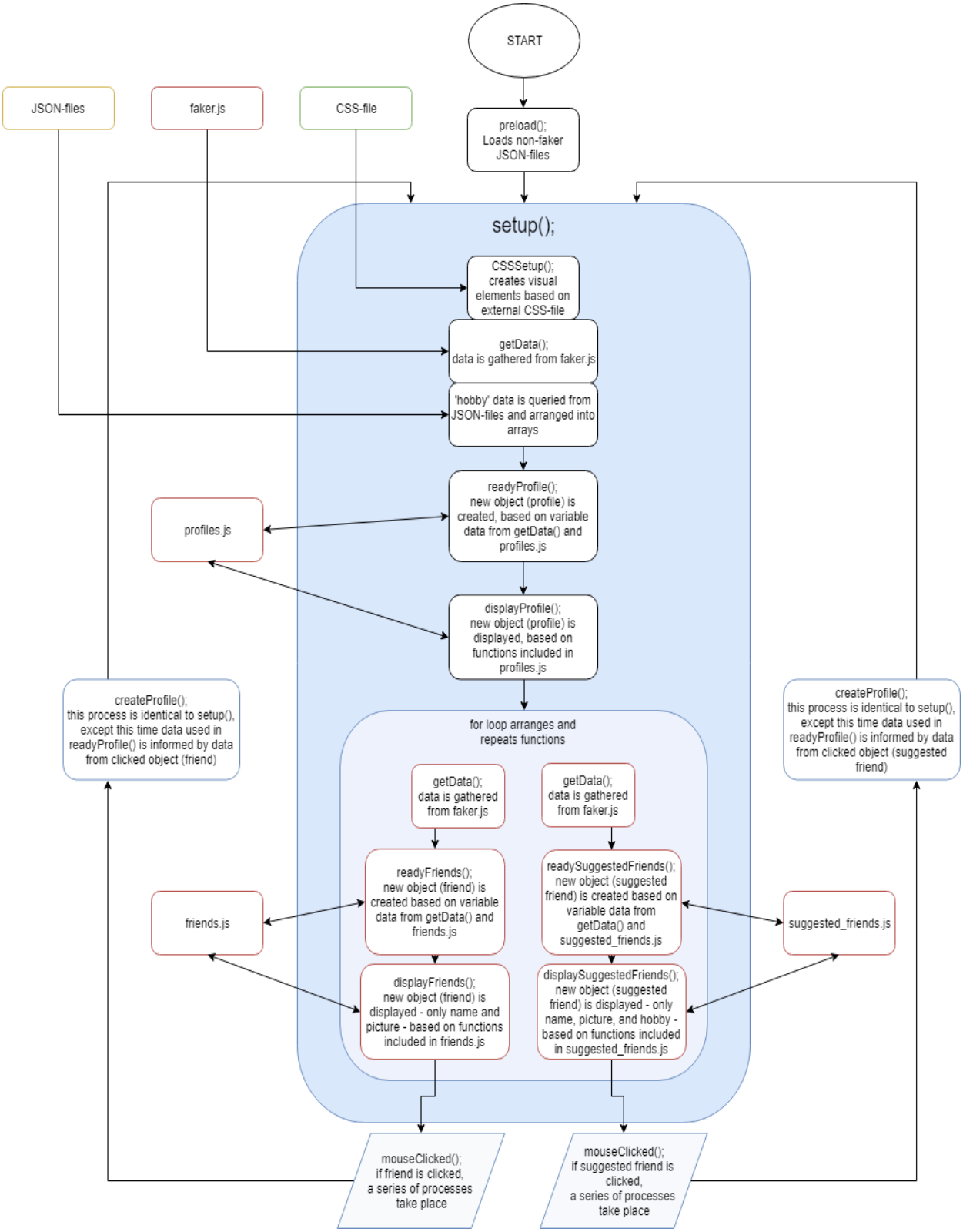
The profile is comprised by a combination of distinct, unconnected strings of data. The data is gathered from an external library called faker.js and JSON-files of our own creation. Every time the program is executed, a new profile is generated. As such, each profile is a unique constellation of separate, but artificially conjoined data. Upon clicking a profile from the 'friends' section, a new profile is generated, though with basis in the information that was already provided (name and picture). The same happens in the case of 'suggested friends', though these are additionally guaranteed to share a common interest with the previous profile. The exact nature of the profile generation will be discussed in greater detail later.

The why

In creating this program, we wanted to explore the concept of online profiles and their relation to object-orientation. What information is considered relevant in creating a profile, what are the representative and interpretative limitations of objectification, and how are these issues reified through the materiality of code? – these are the questions we wanted to, not just examine, but also exemplify through our project. It is an inquiry into the role(s) of abstraction, ambiguity, and absurdity in

computational culture. In what follows, we will attempt to expand on these issues, with reference to selected texts and the program itself.

Flowchart


START
JSON-files
faker.js
CSS-file
preload();
Loads non-faker JSON-files
setup();
CSSSetup();
creates visual elements based on external CSS-file
getData();
data is gathered from faker.js
'hobby' data is queried from JSON-files and arranged into arrays
readyProfile();
new object (profile) is created, based on variable data from getData() and profiles.js
profiles.js
displayProfile();
new object (profile) is displayed, based on functions included in profiles.js
createProfile();
this process is identical to setup(), except this time data used in readyProfile() is informed by data from clicked object (friend)
createProfile();
this process is identical to setup(), except this time data used in readyProfile() is informed by data from clicked object (suggested friend)
for loop arranges and repeats functions
getData();
data is gathered from faker.js
getData();
data is gathered from faker.js
readyFriends();
new object (friend) is created based on variable data from getData() and friends.js
readySuggestedFriends();
new object (suggested friend) is created based on variable data from getData() and suggested_friends.js
friends.js
suggested_friends.js
displayFriends();
new object (friend) is displayed - only name and picture - based on functions included in friends.js
displaySuggestedFriends();
new object (suggested friend) is displayed - only name, picture, and hobby - based on functions included in suggested_friends.js
mouseClicked();
if friend is clicked, a series of processes take place
mouseClicked();
if suggested friend is clicked, a series of processes take place

A technical perspective

Following this brief overview of our project, we now want to dedicate a bit of space to delve deep into the material aspect of our program; that is, the code. By looking closely at a select few parts of the code, we seek to demonstrate how certain themes or ideas can be traced in the code itself. And contrarily, how specific lines of code can serve as the basis for analysis or discussion. Our program is centered around profiles. Profiles, it can be argued, are a way of representing people in the digital realm. Profiles are usually an amalgam of information. It becomes an abstract way of thinking about a person – in isolation, and in relation to others.

In our program, we are literally making people into objects. That is to say, the profile that is supposed to represent a hypothetical person, is actually an instantiation of a class called 'profiles'. The code related to the creation of profiles is primarily found in the profiles.js file. This file is obviously communicating with other parts of the program, but it is no accident that even the very code allowing for creation of profiles, has itself been segregated into its own separate object. An all-important and common function in the code is 'getData'. Here is an excerpt from it:

```
function getData() {

    firstName = faker.name.firstName();

    lastName = faker.name.lastName();

    birthdayMonth = faker.date.month();

    country = faker.address.country();

}
```

The faker.js library is utilized in the gathering of data, and this data is passed on to the creation of profile-objects. The library is itself a collection of distinct object-like JavaScript-files. As such, the 'movement' of our program is one in which strings of data are split into pieces, rearranged, and then ultimately forgotten as the user initiates the creation of new profiles, friends and suggested friends. Each and every object created is unique, or at least a statistical rarity, yet as a new one is created, the old one is removed and no longer accessible. In other words, the profiles have no independent existence outside of the execution of the program.

Our profile-objects are comprised of more than just the four categories shown above. In any case, the categories are arbitrary, insofar as they were dictated by us. It is not at all clear how many more categories would be required to create a greater sense of authenticity or "wholesomeness". As is, the data is randomly gathered from strings of names, locations, and so forth, and fed into the appropriate variables. We find it interesting how the right-hand side of the above code, the part pertaining to the gathering of data, could hypothetically be replaced by some other source – a person actually typing in the information, for example – and very little would change. The computational processes do not require any external verification of the "realness" of data. Such an evaluation must come from elsewhere.

The capacity for creating profiles was one aspect we wanted to explore. The internal dynamic between profiles, fake or not, was another. As such we created different kinds of classes, allowing for different objects. The 'profiles' class can be considered the creator of more detailed objects, owing to the fact that it has the greatest amount of information attached to it. The 'friends' and 'suggested friends', however, are smaller in that sense. Beyond the use of faker.js, there is more sleight-of-hand going on. Here is an excerpt from both object classes:

```
class friends {
  constructor(f_name, l_name, profile_picture) {
    this.f_name = f_name;
    this.l_name = l_name;
    this.profile_picture = profile_picture;
  }
class suggested_friends {
  constructor(f_name, l_name, profile_picture, hobby) {
    this.f_name = f_name;
    this.l_name = l_name;
    this.profile_picture = profile_picture;
    this.hobby = hobby;
  }
```

As can be seen, the two classes are nearly identical, aside from the syntax relating to 'hobby'. Ironically, the 'suggested friend' is then more elaborate than the 'friend'. This runs contrary to the assumption of genuine (and greater) relation between a profile and its "friends". Between the main profile and the suggested friends, there is then a greater dynamic at play. While not necessarily representative of the code behind actual webpages with real profiles, it does show the way, however limited, in which some online profile-objects might be made to interact with each other. The bare-bones approach to these object classes expands on the potential absurdity of the program: However lackluster and rudimentary the main profile may seem, its "friends" are even more simple. Given the guarantee of a common 'hobby' denominator, the relation to the 'suggested friends' may even seem stronger.

A theoretical perspective

We seek to position our work in relation to object-orientation (OO). This topic is inherently related to computational culture and so is aligned with our inquiry. On a general basis – not excluding computational processes – object orientation revolves around the formalization of discrete objects and the constellations within which these conceptual objects interface with one another. Object-oriented analyses treat objects as ontological building blocks, and in doing so, provide an approach to conceptualizing the world as comprised of ontological units of meaning, whether individually distinct or interlocked in overarching systems.

The connection between object-orientation and computational culture stems from the creation and advancement of object-oriented programming (OOP). OOP was formulated in 1961 and is traditionally accredited to Ole-Johan Dahl and Kristen Nygaard. While their contribution to the birth of the object-oriented programming paradigm is evident, OOP has since grown enormously as the result of a complex interplay between ideas, constraints and developers (Black).

Overall, software that follows the principle of OOP can be broken into abstract software artefacts – objects. These objects operate as the basic elements of the software. Objects act as a container of both data, which is characterized as the attributes of the object – and operations, which are defined as the methods of an object (Lee, 18-19). In the case of our program, a profile-object has certain data that can be displayed (name, location) and certain rules dictating possible behavior (interacting with other objects, for example).

Through the OOP paradigm, all of these disparate elements and functions are combined into a single unified object (Lee, 17). For the internal components of the object to be called upon, any message must interface with the object itself, thus actualizing the object as a material component in the structuring of data, as well as manifesting a syntactical hierarchy in the code. This is seen in our program, for example, when data from faker.js is translated into internal object variables. Similarly, when the faker.js library is called upon, the syntax suggests an object-like hierarchy with functions like 'faker.date.month()'.

It would be an understatement to say that OOP has been useful in software development. OOP has cemented itself as a core element of programming culture and practice, effectively reducing the necessary amount of work required on the part of the programmer. In our program, the use of objects was not only essential to the expressive qualities of the program, but also of absolute tool-like necessity for us, as developers with limited programming experience.

However, while lauded as effective and productive, there are also problematic aspects to object-orientation. This is discussed by Cecilie Crutzen and Erna Kotkamp in an essay title *Object Orientation*. They argue that "object-orientation has also become a methodology and theory of interpretation, representation, and analysis of worlds of human interaction with which the computer interfaces […]" (Crutzen and Kotkamp, 201). The problematize the validity of this ontological approach in the discipline of informatics. This resonates well with our ambition to explore the concept of objects as a representative formalization of people, through profiles. Modelling phenomena through code necessitates that any such representation is appropriated to the structural materiality of the software.

In the case of OOP, "[…] a real world phenomenon can only have a representation within the world of artificial objects when it fits into an object class." (Crutzen and Kotkamp, 204). This abstraction of real world phenomena is effectively realized through object-oriented code structures. Abstractions, say Crutzen and Kotkamp: "[…] are simplified descriptions with a limited number of accepted properties. They rely on the suppression of a lot of other aspects of the world." (Crutzen and Kotkamp, 203). Indeed, as we have shown in our code, there is a great degree of arbitrariness at play in our program, in terms of what types of information is considered relevant and how the objects can interact with one another. While our particular code is merely one example of profile-objects, some degree of abstraction and filtration will always be present.

A critical perspective

So far, we have fixated on key parts of the actual code and provided a bit of theoretical background for our project. In the following we will discuss the merits of our program as a critical work, carefully consider certain implications of the program, and speculate on ways in which the program itself proves problematic.

What is a profile?

Our program deals with online profiles. As suggested previously, online profiles are ways of representing people within digital cyberspace. Profiles make up an interesting intersection between culture and technology: a digital token of personhood. We have additionally suggested two important aspects of profiles: they are comprised of (personal) data and are able to interface with other profiles. These were our provisory definitions and, as we eventually discovered through our program, they are not entirely satisfying. That is not to say, that the definition(s) itself is wrong. Rather, the entire concept of profiles is called into question.

Our program, by adhering to an object-oriented structure, exemplifies the limitations of artificially objectifying a person. The stipulated criteria for what constitutes (or in programming terms, *constructs*) a profile, appears arbitrary. Would including more information in our profiles make them any more real? They may come across as more convincing, but surely that is not the same thing. We do not claim to have irreversibly annulled neither the notion nor the potential value of profiles, but we do believe that we have successfully used our program to ask difficult questions.

Fake does not matter

The data included in the faker.js library is a mixture of real pictures, fictitious places and made-up numbers. That data has been encapsulated under the title of being fake. Fakeness suggests an opposite, a kind of 'true' data, whatever that means. The constellation of data created by our program is certainly artificial, but then, that could be argued to be the case with all software. We do not know under what circumstances that data was 'captured' - if the names were made up by the same person, gathered from yet another list, and so on. Ultimately, though, that data was introduced via a keyboard (or similar), somewhere, at some point in time.

In translating or formalizing phenomena into data, the original source becomes, computationally speaking, irrelevant. Whether gathered through APIs, libraries or keyboard input, the data is treated in the exact same way, as demonstrated in the getData-function. This had quite an impact on us, and while our focus here is not on data capture, we do believe that our program demonstrates the sterile way in

which data, whatever its source, is processed, once internalized by the program. In a similar way, the profile-objects themselves are treated as coherent units. The issue of 'realness' is never brought up in the underlying processes.

The dynamics of objects

We wanted to create a program in which objects can act upon other objects. In our case, a material connection is made between profiles and 'suggested friends', based on common hobbies. While not representative of equivalent code on websites with real profiles, this does show how select aspects of objects can be made to interface with others. In critical terms, it shows how the process of objectification can turn into *commodification*. Not just friends, but advertisements, websites, groups or similar offers, whether political or economic in nature, could be suggested to a given profile, merely based on a hobby, location or name.

As creators, we had the privilege of determining not only the parameters constituting a profile, but also the ways in which it might interact or inform other profiles. In most other cases, this degree of transparency is not available. In any case, our process of creating the program highlights a digital hierarchy of power and information. In formalizing the profile into a class structure, it is not so much the person being represented *by* the profile, but the creator *of* the profile (class), that has the power to define what is or is not relevant, allowed, meaningful or appropriate.

By turning people into profiles and situating the profiles within the boundaries of object-oriented programming, the object class becomes the determinant of what it means to be a person – at least on the internet. If every single person is expected to fit into the 'mold' of a profile-object, important details and differences are bound to be lost, not just in translation, but in assimilation. As profiles are abstractions, not just of individuals, but of people in general, they become powerful tools for 'streamlining' and literally processing individuals into an abstract whole, however erroneous or problematic (Crutzen and Kotkamp, 203-204).

A fun exploration

While our overarching theme and focus has been on object-orientation, we would be remiss to ignore the value of actually experiencing the program. As strings of unrelated data are put into proximity of one another, frankensteinian pseudo-profiles appear in a sort of 'gestalt'. On numerous occasions, we were amused or puzzled by the absurd profiles constructed by our program. While purely anecdotal, this experience serves a testimony to the compelling power of proximity: arrange certain pieces of data

together, and people might believe them to be related. In a sense, it is also a cautionary tale to warn against the ease with which faux profiles or similarly bogus content can be created and shared.

At the same time, it reveals certain biases at play. As much as we can laugh at the program, it also asks us to question our own assumptions of what makes something believable or authentic. In working on this program, we would often debate whether the profiles should be made more overtly absurd or implausible. However, we never managed to settle on a satisfying 'standard' for what would be considered normal. This either contradicts or reinforces a previous point: profiles seem like lackluster representations of people, yet we are all too willing to use them as a foundation for passing judgment on others.

Black Boxes

The use of faker.js is a topic in its own right. There are significant issues to address. Some of them technical, others political, and others yet, ethical. In carefully considering and reflecting upon our own code, we have sought a greater understanding and transparency of the interplay between objects. In using an external source of data, we potentially run the risk of substituting one black box with another. While we have adapted the particular syntax related to the faker.js library, there is still a significant degree of opacity left with regards to the inner workings of the library: how the data was produced, in what way it is distributed, and hypothetically, any prior connection between the data. It is not within the scope, nor ambition of this project to fully expand and satisfyingly address these concerns, but we do wish to acknowledge their reality and significance.

Sources

Black, Andrew P. "Object-oriented Programming: some history, and challenges for the next fifty years", 2 Mar 2013

Crutzen, Cecile and Kotkamp, Erna. "Object Orientation", in Software Studies\a lexicon, Eds Matthew F. MIT Press, 2008. 200-207

Lee, Roger Y. *Software Engineering: A Hands-On Approach*. Springer, 2013. 17-38