Design Studio: Algorithms as Decision Makers

Created by: Evan Peck, Bucknell University
To be completed in lab. Groups of 3

In this lab, we will create algorithms that determine housing.

When we say the word 'algorithm', we tend to ascribe agency to the computer. It is *deciding* things for us. But the reality is that there is no magic. There are software developers like you and me who design and create sets of rules that the computer carries out for us.

These algorithms are all around us, and they are *constantly* making decisions.

The decisions we make in code impact the lives of real people. For example, the **Silicon Valley Triage Tool** is an *algorithm* that identifies homeless people for whom giving them housing would cost the public less than keeping them homeless. So even as we learn the simple structures of code, we need to think about *how can we make **good** decisions?* When the livelihood of people depend on us, *how can we be fair?*

We are going to explore this idea in a more familiar context to you - **university housing**. Universities like Bucknell select methods that determine the order in which students can choose their housing. You might not think of it as one, but this method is **an algorithm.** In this studio, you will have the opportunity to design your *own* algorithm. We're also going to begin dabbling with a human-centered design process to make sure that **the decisions we make are never untethered from the people we impact.**

In this studio, you'll practice...

- Translating English rule-sets into code
- Soliciting text input from people
- ***Applying conditionals (if elif else) to make decisions with a program***
- The application of an accumulation variable to keep track of information in a program
- Integrating basic human-centered design processes into your programs

The Decision-Maker: Who gets to choose their housing first?

Your job is to build an algorithm that helps determine the order in which students will get to select their housing. To simplify things, we're going to use a *point system*.

- Students are awarded a number of points based on a variety of factors.
- Students with the most amount of points get first choice at housing.

This real approach is used by many universities. For example, consider the following *real point system* used by another liberal arts college in the United States:

- *Current Freshman:* 1 point
- *Current Sophomore:* 2 points
- *Current Junior:* 3 points
- *Current Senior:* 4 points
- *23+ Years of Age:* 1 point
- *Full-Time, Off-Campus Program Credit (e.g. student teaching):* 1 point
- *Academic Probation:* -1 point
- *Possible Academic Suspension:* -2 points
- *On Disciplinary Probation at Any Point during the Academic Year:* -3 points

So a junior (3 points) who is 23 years old (1 point) would have priority over a senior (4 points) who is on academic probation (-1 point).

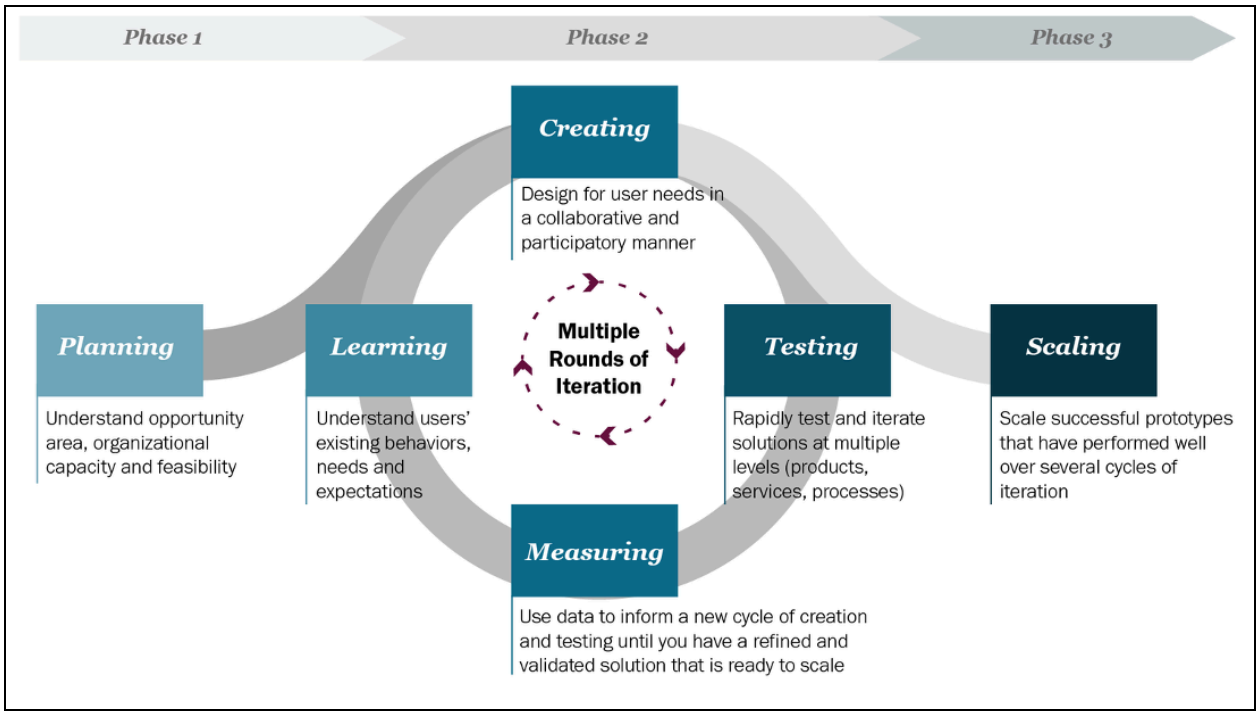
Your goal: Create a program that assigns points to students in order to prioritize them in housing selection. But wait! Don't start yet. First…

Before you Code, Assess the Needs of Your Users

While the list above was *one* college's take, there are **many** more potential aspects to consider if you want to create a **fair** algorithm that takes into account the diverse needs of students.

You should not create a program that serves people without talking to people. Talk to other students in class. Ask them about their needs. What other unique factors may be important in deciding who should choose housing first?

Your goal: Collaborate on a bullet point list of factors that came up in conversation. Give each one a numeric priority.
(You should include your bullet point list in your colaboratory report)



Talking to users, prototyping ideas, and then testing them with users, is part of what is known as the **human-centered design process**… a key process for developing useful and usable programs

Design and Plan Your Point-Assigning Algorithm

Now it is time to translate our student needs into a concrete algorithm. Be careful and limit yourself to the most important factors, *you have a limited amount of time in this class!*

Our algorithm will:

1. Ask students questions (like *What class year are you?*)
2. Assign points based on their answers (like *4 points for senior*)
3. Accumulate their total points across all answers (like *You have 23 housing points*)

Constraints: Since we are exercising our ability to write code in addition to solving problems, we are going to put a couple of constraints to make sure you get practice with the different ways in which conditionals can be used.

- You must have at least 1 question that *only* appears if the previous question is answered a specific way.
    - *For example:* if someone says they are a 4th year student, you may ask the question "Are you about to graduate?" If they say yes, they receive no points. *ONLY* a 4th year student would receive this question.
    - ***Hint:*** Use nested if statements.

- You must have at least 1 question that makes use of a simple equation to determine the number of points awarded.
    - *For example:* Rather than ask people to enter their status as a 1st, 2nd, 3rd, 4th year, you ask them for the number of credits they have received so far at Bucknell. Then they might receive `credits/8` points.
    - ***Hint:*** Remember that the `input()` statement returns a string. You'll need to turn it into an integer.

Your goal: Create a bullet point list that describes the factors you are considering, and how you are mapping those factors to points.

Before you Code, Make Sure it Works: User Testing

You should not create a program that serves people without testing it on people. At a minimum, you should test it with a couple of people around you.

- Were the results what you expect?
- Did you discover any cases which you haven't accounted for previously?

Your goal: Write **at least** 3-5 hypothetical *test* cases for your program. For example:

- A 25 year old senior who is on academic probation should output 4 points
- A 22 year old junior who is student teaching should output 4 points
- A 20 year old sophomore on disciplinary probation should output -1 points

TA / Instructor Check 1

Check that the team sufficiently explored the needs of *different* people around them.
Check that the algorithm fulfills the requirements above.
Check that the scope is appropriate for the time.
Check that the test cases are appropriate.

Write Code that Automates your Decision-Making Process

Now it's time to translate your algorithm into code.

Below, I've included a program that you can start out with. It uses the example point system above to account for a student's class year and age at the university. Copy-and-paste it into your editor. Run it to make sure you understand what's happening.

```
# Created by: Evan Peck (Bucknell University)
# - Contact: evan.peck@bucknell.edu
```

```
# - Last Modified: April 6, 2019

# Keeps track of the point total during questions
current_score = 0

# A header to start the program
print("--------------------------------")
print("   HOUSING SCORE CALCULATOR")
print("--------------------------------")
print()

# Assign points based on class year
print("QUESTION 1")
year_ans = input("What year are you? (1, 2, 3, 4): ")

if year_ans == "1":
    current_score += 1
elif year_ans == "2":
    current_score += 2
elif year_ans == "3":
    current_score += 3
elif year_ans == "4":
    current_score += 4

# If the student is >= 23 years old, give them another point
years_old = input("How old are you?: ")

if int(years_old) >= 23:
    current_score += 1

# At the end of the program, tell the user their score
print()
print("------YOUR HOUSING SCORE----------")
print("Your housing points score is", current_score)
print("---------------------------------")
```

Your goal: Now modify the program to match *the algorithm you designed*. During your creation, keep a couple of things in mind:

- Use comments to describe what was happening in the program
- Choose variable names that clearly describe that data that they hold.
- Use spacing to group similar code.

Is it correct?:

- You should check your code with the test cases you outlined above. In a future lab, we'll

discover ways to do this automatically, but for now, the important piece is that you are **ensuring that your code is correct.**

TA / Instructor Check 2

Check for functionality - does the code work, do the tests work, does it meet the specifications?
Check for style - how can the style be improved?

Your Code Works... but is it fair? (Individual)

You should never deploy real code without checking your assumptions.
Your test cases tested your *technical* assumptions, but not your *not your social* assumptions

1. Find classmates or students outside the class (there are often students in the lobby)
2. Run your code with them
3. Get feedback on what worked and what didn't?

In particular, you should reflect on...

1. **Which students are most likely to benefit from your algorithm?**
2. **Which students are most likely to be forgotten by your algorithm?**

You should also **read a reflection** about a more complex (but in many ways similar) scoring system:

- *We created poverty. Algorithms won't make that go away.*
  By Virginia Eubanks

Finally write your group's reflections in your report

Before tomorrow... (Individual)

- **Each member of the team should print out a copy of the CODE and bring it to your lecture tomorrow.**
- Following this lab, you should come up with two good questions about python coding. See our Google classroom page to fill out your questions.
- **Additional reading:** *What Happens When An Algorithm Cuts Your Health Care*
  By Colin Lecher