Algorithms

Lesson Duration: 60 minutes

Learning Objectives

- Understand what an algorithm is
- Be able to apply code to solve problems

What is an Algorithm?

Likely, no matter your background, you've heard or read the word algorithm being thrown around, especially in Hollywood, or in commentary on big tech companies and the ethics of how they use them to analyse and monitize user data and behavior. But, what exactly is an algorithm?

One definition Webster offers for an algorithm is "a step-by-step procedure for solving a problem or accomplishing some end". These procedures are reusable. Does that sound like any code construct you've worked with so far?

> Think of finding out how long a list is, or how to add/remove items from them

Yes, functions! A function, or method, is essentially an algorithm implementation. Algorithms can be written to handle virtually any computational task, and their purposes range from simple calculations to data mining and analysis, advanced AI, and everything in between.

In today's session we will show you a couple of ways to solve certain problems!

Our first algorithm

If you are excited about debunking and working with some algorithms, I have great news for you - if you submitted any of your previous tasks from the first 2 weeks, you have already created algorithms - on your own, no less! Congratulations!

Consider this: On week 1, our task was to handle input from a user, depending on their choices, modify a value, then display results.

Using conditional statements, functions to collect input from the user, and other functions to

print end results back to our user would certainly fall under the definition of an algorithm!

Copy the following into a new file called `algorithms.py`

```
pin = 1234
balance = 50000

input_pin = input("Please enter your PIN number!")

if(pin == int(input_pin)):
    print("Your balance is " + str(balance))
else:
    print("Incorrect PIN! Please try again!")
```

You were presented with a problem, you wrote step-by-step instructions that, when executed, solved our problem!

As with any good algorithm, of course, this could use improvement! A common way of tackling problems is the classic `red-green-refactor`.

1. Have a problem at hand - `red`
2. Solve the problem - `green`
3. Improve the solution to make it faster/easier to read/more efficient/more featureful - `refactor`

Refactoring means that our current code meets expectations and solves problems. Having said that, let's take a look how can we improve our our week 1 task!

Consider the following cases! - What happens if we want to limit the number of guesses to 3? - What happens if the user is entering something that cannot be turned into an `int`?

Knowing what we know after week 2, lets build a better solution to these problems!

First, we are going to tackle the limit to 3 tries. One of the ways we can do this is to keep track of the tries, and every time the user makes a mistake, we reduce the number of tries and ask again.

What construct could help us repeat code `while` we have remaining tries?

```
pin = 1234
balance = 50000
number_of_tries = 3

input_pin = input("Please enter your PIN number!")

while(number_of_tries > 0):
    if(pin == int(input_pin)):
        print("Your balance is " + str(balance))
    else:
        input_pin = input("Incorrect PIN! Please try again!")
        number_of_tries -= 1
```

With these changes, we could keep track of how many tries we have left, and "lock" the user out once they reach the limit.

But wait a second - if we actually guess the PIN correctly, we display the balance. But now we get stuck in an infite loop, because we don't reduce the number of tries remaining, so we break our logic by not providing an exit condition.

> If you're stuck in an infinite loop, remember to click in the terminal and press `CTRL` + `C` (both operating systems)

How can we figure out a solution for this?

> Search for `how to exit from while loops in python` online

Searching is an essential skill that every developer needs to get better at. Most things exist in blogposts, documentation, or the internets largest forum for devs - `Stackoverflow`

> Take a look at results, read through the first couple of posts, and try to figure out how to exit from `while` loops.

After digging around a bit, we can see that the easiest solution would be to use a break statement after the condition of correctly guessing the PIN is met!

```
pin = 1234
balance = 50000
number_of_tries = 3

input_pin = input("Please enter your PIN number!")

while(number_of_tries > 0):
    if(pin == int(input_pin)):
        print("Your balance is " + str(balance))
        break # ADDED
    else:
        input_pin = input("Incorrect PIN! Please try again!")
        number_of_tries -= 1
```

> Task: Try to solve the second improvement! What happens if the user is entering something that cannot be turned into an `int`?

Example solution:

```
pin = 1234
balance = 50000
number_of_tries = 3

input_pin = input("Please enter your PIN number!")

while(number_of_tries > 0):
    # .isdigit() checks if a string is valid as an int
    # not turns the conditional check to the oppositne. .isdigit() would
    if(not pin.isdigit()):
        print("Please enter a valid PIN number")
        break
    if(pin == int(input_pin)):
        print("Your balance is " + str(balance))
        break # ADDED
    else:
        input_pin = input("Incorrect PIN! Please try again!")
        number_of_tries -= 1
```

Now that we saw how can we solve some problems, its time to look around for some more advanced problems!

Consider the following dataset (copy it to `algorithms.py`):

```
animals = [
    {"name": "capybara", "group": "mammal", "weight": 110},
    {"name": "hedgehog", "group": "mammal", "weight": 0.5},
    {"name": "bearded dragon", "group": "reptile", "weight": 1},
    {"name": "tortoise", "group": "reptile", "weight": 40},
    {"name": "hummingbird", "group": "bird", "weight": 0.01},
    {"name": "elephant", "group": "mammal", "weight": 10000},
    {"name": "ostrich", "group": "bird", "weight": 280},
    {"name": "python", "group": "reptile", "weight": 180},
    {"name": "blue whale", "group": "mammal", "weight": 300000},
    {"name": "lion", "group": "mammal", "weight": 350}
]
```

Let's consider the following tasks:

1. Print out all the animals names that are heavier than 100 pounds!
2. Print out the heaviest animal!
3. Print out the lightest animal!
4. Print out all mammals as a list!

If you feel confident, give it a try on your own, then continue for the solutions. Otherwise, let's think about these!

The first task is a classic problem we've seen last time - we need to check each item in a list, and if they meet a condition, we need to print out their details.

Let's write some pseudocode for this:

```
# Loop through each animal
# check their weight value
# if their "weight" value is more than 100, print out the animal's name
```

Let's follow our steps to solve this problem.

```
for animal in animals:
    if animal["weight"] > 100:
        print animal["name"]
```

Keep in mind we started with no code, got a problem on our hand and we wrote code to solve the issue!

The next problem is considerably harder to write - unless we take the pseudocode approach!

```
# loop through all animals
# compare our animals to each other animal
# whichever animal is heaviest, we should print out their name
```

Solving the task in your head is 3/4 of the battle already won. Now we just need to write the code and consider a couple of things.

We probably have an idea on how to compare values - if one animal outweighs another, its heavier.

But what do we compare our first item to? And once we did, how are we keeping track of the heaviest animal?

We could declare the first item the heaviest (its true too - until we check any other, the first animal is the only one we checked, thus technically the heaviest) and start comparing from there. Once we find a heavier animal, we replace our value!

```
heaviest_animal = animals[0]

for animal in animals:
    if animal["weight"] > heaviest_animal["weight"]:
        heaviest_animal = animal

print(heaviest_animal)
```

Fantastic!

Solving for the lightest shouldn't pose a challenge now!

```
lightest_animal = animals[0]

for animal in animals:
    if animal["weight"] < lightest_animal["weight"]:
        lightest_animal = animal

print(lightest_animal)
```

Solved!

The important thing is this: We were able to solve this problem faster *because we've seen the pattern of how to solve something similar first*.

Humans are fantastic at pattern recognition, so its always easier to try to look up solutions first, and then when presented with a new, custom task, we can use our previous experience to recognise patterns and solve issues!

The last task is a bit tricky:

- Print out all mammals as a list!

The problem could be fairly easy to solve with a simple condition - however we need to print a list, not just the animals one by one. This could be useful if we wanted to further refine our filtered results.

```
# Loop through all animals
# if the animal's group is mammal, add it to a list
# once finished, print out the final results
```

Just by writing out our problem, we might've come to a conclusion - we need an empty list to act as our filtered list for values!

```
mammals = []

for animal in animals:
    if animal["group"] == "mammal":
        mammals.append(animal)

print(mammals)
```

Job's done!

The great news is, with these problems being solved with algorithms, you can refer back to them once something else comes up as a task - if I were to ask you to create a list of all animals that has more than 5 letters in its name, you could easily do that, because part of the problem is already solved in a previous exmaple - you just have to focus on the unknown parts of the question!

Conclusion

We've discussed what algorithms are and how can we use some simple ones to solve certain problems. We've seen that a lot of problems have similar patterns to them, a solution to a problem can be used in other tasks.