String Manipulation

In this lecture we will talk about:

- String Manipulation
  - charAt(int)
  - indexOf(String or char)
  - substring(int, int)
  - replace(String, String)
- Exploiting Patterns

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:
  - Computer science has a number of concentrations.
    - General concentration is the most flexible, and even allows students to double major or minor pretty easily.
    - Software Engineering
    - Computing Systems
    - Human Centered Computing
    - Networks and Security
    - Artificial Intelligence
    - Computer Science Education.
  - Minors:
    - Minor in Computer Science - choose your own adventure minor
    - Minor in Machine Learning - popular with stats/math, and engineering
    - Minor in Bioinformatics - Biology + Computer Science

Reading Check in

Given the following code, what would be the output?

In [7]:
```java
String plant = "kinnikinnik";
int loc = plant.indexOf("k", plant.indexOf("k")+1);
String out = plant.substring(loc, loc+3);

System.out.println(out);
```
```
kin
```

String

- It has data
- It has functionality (methods)
- It is also immutable (can't be directly modified)
  - Every method that builds a String, returns a copy
  - Java does this for memory efficiency
- String are a sequence of characters, with indices

Index	Character
0	k
1	i
2	n
3	n
4	i
5	k
6	i
7	n
8	n
9	i
10	k

In [12]:
```java
String plant = "kinnikinnik";

System.out.println(plant.charAt(5));
System.out.println(plant.charAt(plant.length()-1));

for(int i = 0; i < plant.length(); i++) {
    System.out.print(plant.charAt(i) + "_");
}
```
```
k
k
k_i_n_n_i_k_i_n_n_i_k_
```

In Class Activity

Complete the following method.

The goal is for it to return the index of the character it finds in the String str.

- Start your loop at the `start` parameter.
    - hint: `int i = start`

In [15]:
```java
public static int find(String str, char c) {
    return find(str, c, 0); // overloaded method!
}

public static int find(String str, char c, int start) {
    for(int i = start; i < str.length(); i++) {
        if(str.charAt(i) == c) {
            return i;
        }
    }

    return -1; // discussion item: why do we opt for minus 1 if we don't find it?
}
String plant = "kinnikinnik";
System.out.println(find(plant, 'k'));
System.out.println(find(plant, 'i', 5));
System.out.println(find(plant, 'J'));
```

```
0
6
-1
```

Common String Methods

- `.charAt(int)` - gives us the character at location
- `.indexOf(char)` - gives us the location of character (what you just wrote!)
- `.indexOf(String)` - overloaded option, gives the location of the *start* of the string that matches
- `.indexOf(char, int)` or `.indexOf(String, int)` - same as above, but changes starting location
- `.lastIndexOf(char)` - gives us the index starting at the end working down (also has String version)
- `.substring(int start, int end)` - returns the substring from start - including start, to end, excluding end. (inclusive/exclusive)
- `.toLowerCase()` - returns the lowercase version of the String
- `.toUpperCase()` - returns the uppercase version of the String

In [19]:
```java
String latin = "SATOROTAS";

System.out.println(latin.indexOf("T"));
System.out.println(latin.indexOf("TOROT") + " notice it is the 'start' location of the
System.out.println(latin.indexOf("T", latin.indexOf("T")+1)); // why did I need the +
```

```
2
2 notice it is the 'start' location of the other String
6
```

In [22]:
```java
String sub = latin.substring(0, latin.indexOf("R"));
```

```
System.out.println(sub); // what will this print?
```

```
SATO
```

In [23]:
```
String sub = latin.substring(latin.indexOf("R"));
System.out.println(sub); // what will this print?
```

```
ROTAS
```

Exploiting Patterns

Strings often have patterns we follow, and these methods help us exploit them.

For example:

```
<html><body><h1>Heading</h1><p>This is my cool interesting paragraph</p>
</body></html>
```

The pattern above uses < / > style notation! By seeing that pattern, I can build webpages! (really, that is html code for webpages).

Another pattern to think about:

```
Fort Collins,40°35'6.9288"N,105°5'3.9084"W
Denver,39°44'31.3548"N,104°59'29.5116"W
Boulder,40°0'53.9424"N,105°16'13.9656"W
```

The values are different, but there is a pattern in each string

```
CITY,LATITUDE,LONGITUDE + direction as the last character
```

Often writing down the pattern is valuable in figuring out the problem!

In [2]:
```
String coord = "Fort Collins,40°35'6.9288\"N,105°5'3.9084\"W"; // the \ lets me keep t

String city = coord.substring(0, coord.indexOf(","));
System.out.println(city);
```

```
Fort Collins
```

In class activity

Complete the `Location(String)` constructor in your in class activity - `Location.java`.

- You will use substring + indexOf - to break up the String above into its **three** parts.
- Make sure to check for OB1 errors!

In [10]:
```
class Location {
    private String name;
    private String latitude;
    private String longitude;

    public Location(String name, String latitude, String longitude) {
        setName(name); // let the methods handle the set incase they want to modify a
        setLatitude(latitude);
        setLongitude(longitude);
    }

    public Location(String location) {
        String name = location.substring(0, location.indexOf(","));
        String longitude = location.substring(location.lastIndexOf(",")+1);
        String latitude = location.substring(location.indexOf(",")+1,
                                             location.lastIndexOf(","));
        setName(name);
        setLatitude(latitude);
        setLongitude(longitude);
    }

    public void setName(String name) { this.name = name;}
    public void setLongitude(String longitude) { this.longitude = longitude.toUpperCas
    public void setLatitude(String latitude) { this.latitude = latitude.toUpperCase();

    public String toString() {
        return String.format("{name:%s, lat:%s, lon:%s}", name, latitude, longitude);
    }


}


ArrayList<Location> locations = new ArrayList<>();
locations.add(new Location("Fort Collins,40°35'6.9288\"N,105°5'3.9084\"W"));
locations.add(new Location("Denver,39°44'31.3548\"N,104°59'29.5116\"W"));
locations.add(new Location("Boulder,40°0'53.9424\"n,105°16'13.9656\"w"));

System.out.println(locations);
```

```
[{name:Fort Collins, lat:40°35'6.9288"N, lon:105°5'3.9084"W}, {name:Denver, lat:39°4
4'31.3548"N, lon:104°59'29.5116"W}, {name:Boulder, lat:40°0'53.9424"N, lon:105°16'13.
9656"W}]
```

Thinking further

- Wouldn't it be great if we could read in all the locations from a file?
  - We learn how to do that on Monday
- The format [{},{}] is actually JSON string format, another 'pattern' that is often used in programming to transfer data.