A map is a set of pairs $\{(x, f(x))\}$.

Each $x$ in domain maps to some $f(x)$ in codomain.
Math examples

Calculus 1 and 2 ("functions"): 

\[ f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m \]
Maps in everyday life

\[ \text{map}(x,y) : \text{position in 2D image} \rightarrow \text{2D position in Montreal} \]
vote_result : US_state ➔ { blue(Dem), red(Rep) }
## Restaurant Menu

<table>
<thead>
<tr>
<th>dish_name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poulet au cari vert &amp; lait de coco \</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken in green curry &amp; coconut milk</td>
<td></td>
</tr>
<tr>
<td>Poulet au cari jaune &amp; lait de coco</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken in yellow curry &amp; coconut milk</td>
<td></td>
</tr>
<tr>
<td>Poulet au cari paneang &amp; lait de coco</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken in paneang curry &amp; coconut milk</td>
<td></td>
</tr>
<tr>
<td>Poulet sauté aux oignons et piments forts</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken sautéed with onions &amp; chillies</td>
<td></td>
</tr>
<tr>
<td>Poulet sauté au basilic thailandais</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken sautéed with thai basil</td>
<td></td>
</tr>
<tr>
<td>Poulet sauté aux noix de cajou</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken sautéed with cashew nuts</td>
<td></td>
</tr>
<tr>
<td>Poulet sauté aux aubergines</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken sautéed with eggplants</td>
<td></td>
</tr>
<tr>
<td>Poulet sauté aux haricots verts</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken sautéed with green beans</td>
<td></td>
</tr>
<tr>
<td>Poulet sauté aux pousses de bambou</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken sautéed with bamboo shoots</td>
<td></td>
</tr>
<tr>
<td>Poulet sauté au brocoli &amp; sauce aux huitres</td>
<td>18.95</td>
</tr>
<tr>
<td>Chicken sautéed with broccoli &amp; oyster sauce</td>
<td></td>
</tr>
</tbody>
</table>
Train Schedule

Schedule: station → time of next train (or list of times)
Index in a book

index : term \rightarrow list of pages containing term
A map is a set of (key, value) pairs. For each key, there is at most one value.
Map (ADT)

Two keys can map to the same value.
Map (ADT)

It is NOT allowed that one key maps to two different values.
The above example is NOT a map.
Map Entry

Each (key, value) pair is called an entry.

In this example, there are four entries.

The black dots here indicate keys or values that are not in the map.
In COMP 250 this semester, the above mapping has ~600 entries.

Most McGill students are not taking COMP 250 this semester.

BTW, the student ID can also be part of the student record.
Map ADT

• put(key, value)

• get(key)

• remove(key)

• ...

Map ADT

• put(key, value)

If the map previously contained a mapping for the key, then the old value is replaced by the specified value, and the previous value is returned. Otherwise, return null.

• get(key)

• remove(key)

• ...

Map ADT

- put(key, value)

- get(key)
  Returns the value to which the specified key is mapped, or return null if this map contains no entry for the key.

- remove(key)

- ...

Map ADT

• `put(key, value)`

• `get(key)`

• `remove(key)`  
  Removes the entry for the key, if it is present, and returns the value. Returns null if the map contains no mapping for the key.

• ...
About the figures....

When programming with maps in Java, keys and value variables are *reference* types. On this slide, the keys as different sized red disks and the values are blue shapes.

In this example, two of the keys map to the same value.
About the figures....

For the remaining slides today, I will draw a set of *(key, value)* pairs, i.e. entries, as shown below.
But try to keep the previous slide in mind...
Data Structures for Maps?

How to organize a set of \((\text{key}, \text{value})\) pairs, i.e. entries?
Array list

How would you implement these operations?
What are the best and worst case time complexities?
Singly (or Doubly) linked list

How would you implement these operations?
What are the best and worst case time complexities?
Special case #1: what if keys are *comparable*?

Can we take advantage of this?
Array list (sorted by key)

How would you implement these operations?
What are the best and worst case time complexities?
When I presented BSTs, I only mentioned the keys. The nodes could instead store key/value pairs and the BST algorithms would still work fine.

How would you implement these operations?
What are the best and worst case time complexities?
minHeap (priority defined by key)

put(key, value)  get(key)  remove(key)

How would you implement these operations?
What are the best and worst case time complexities?
Special case #1: what if keys are comparable?

Special case #2: what if keys are positive integers in a small range?

Then, we could use an array with elements of type $V$ (value) and have $O(1)$ access.

This would not work well if keys are 9 digit student IDs. Why not?
Special case #1: what if keys are *comparable*?

Special case #2: what if keys are positive integers in small range?

General case. What if keys are some other type?

We will define a map from keys to a *large* range of positive integers. Such a map is called a *hash code*.

Next we will look at Java’s `hashCode()` method.

Then, next lecture, I will tell you how to use this hash code.
Recall lecture 13:
Object.hashCode()

class Object

+ Object()

+ equals( Object ) : boolean
# clone( ) : Object
+ hashCode( ) : int
+ toString( ) : String

Returns a (positive) integer.

You can think of it as the address of the object, although this is not required in any technical sense.
**Object.hashCode()**

![Diagram showing 1-to-1 mapping](image)

- Objects in a Java program (runtime)
- Object's address in JVM memory (24 bits)

If `obj1` and `obj2` are reference variables, and if the objects that they reference inherit the `Object.hashCode()` method, then `obj1.hashCode() == obj2.hashCode()` is equivalent to `obj1 == obj2`. 
String.hashCode()

How is `String.hashCode()` defined?
Example of a simpler hash code for strings

\( h(s) \equiv \sum_{i=0}^{s.length-1} s[i] \)  

is the first character in the sequence, \( s[1] \) is second, etc.

e.g. \( h("eat") = h("ate") = h("tea") \)

ASCII values of ‘a’, ‘e’, ‘t’ are 97, 101, 116.
String.hashCode()

\[ s.\text{hashCode}() \equiv \sum_{i=0}^{s.\text{length}-1} s[i] \times x^{s.\text{length}-1-i} \]

where \( x = 31 \).

e.g. \( s = \text{“eat”} \), \( s.\text{hashCode}() = 101 \times 31^2 + 97 \times 31 + 116 \)

<table>
<thead>
<tr>
<th>‘e’</th>
<th>‘a’</th>
<th>‘t’</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[0]</td>
<td>s[1]</td>
<td>s[2]</td>
</tr>
</tbody>
</table>

s.length = 3
String.hashCode()

\[ s.\text{hashCode}() \equiv \sum_{i=0}^{s.\text{length}-1} s[i] \times x^{s.\text{length}-1-i} \]

where \( x = 31 \).

e.g. \( s = \text{“ate”}, \quad s.\text{hashCode}() = 97 \times 31^2 + 116 \times 31 + 101 \)

\begin{tabular}{c}
\hline
\text{‘a’} & \text{‘t’} & \text{‘e’} \\
\hline
\end{tabular}

\begin{tabular}{c}
\hline
\text{s.length = 3} \\
\hline
\end{tabular}

\begin{tabular}{ccc}
\hline
\text{s[0]} & \text{s[1]} & \text{s[2]} \\
\hline
\end{tabular}
public int hashCode()

Returns a hash code for this string. The hash code for a String object is computed as

\[ s[0] \times 31^{(n-1)} + s[1] \times 31^{(n-2)} + \ldots + s[n-1] \]

using int arithmetic, where \( s[i] \) is the \( i \)th character of the string, \( n \) is the length of the string, and \( ^{\text{\scriptsize ^{}}} \) indicates exponentiation. (The hash value of the empty string is zero.)

Overrides:
hashCode in class Object

Returns:
a hash code value for this object.
String.hashCode()

\[ s.\text{hashCode}() \equiv \sum_{i=0}^{s.\text{length}-1} s[i] \times (31)^{s.\text{length} - 1 - i} \]

Q: If \( s1.\text{hashCode}() == s2.\text{hashCode}() \)
then can we conclude \( s1.\text{equals}(s2) \) is true?

A: No. \( s1.\text{equals}(s2) \) may be either true or false.

\[ s1.\text{hashCode}() == s2.\text{hashCode}() \text{ is true, but } s1.\text{equals}(s2) \text{ is false} \]
String.hashCode()

\[
    s\.hashCode() = \sum_{i=0}^{s.length-1} s[i] \times (31)^{s.length - 1 - i}
\]

Q: If \(s1\.hashCode() \neq s2\.hashCode()\) then what can we conclude about \(s1\.equals(s2)\) ?

A: \(s1\.equals(s2)\) is false.
ASIDE: Java uses “Horner’s rule” for efficient polynomial evaluation

\[ s[0] \times 31^3 + s[1] \times 31^2 + s[2] \times 31 + s[3] \]

There is no need to compute each \( x^i \) separately.
ASIDE: Java uses “Horner’s rule” for efficient polynomial evaluation

\[ \text{s}[0] \times 31^3 + \text{s}[1] \times 31^2 + \text{s}[2] \times 31 + \text{s}[3] \]

\[ = ( \text{s}[0] \times 31^2 + \text{s}[1] \times 31 + \text{s}[2] ) \times 31 + \text{s}[3] \]

\[ = ( ( \text{s}[0] \times 31 + \text{s}[1] ) \times 31 + \text{s}[2] ) \times 31 + \text{s}[3] \]

\[
\begin{align*}
\text{h} & = 0 \\
\text{for (i = 0; i < s.length; i++)} & \\
\text{h} & = \text{h} \times 31 + \text{s}[i]
\end{align*}
\]

For a degree \( n \) polynomial, Horner’s rule uses \( O(n) \) multiplications, not \( O(n^2) \).
### Lectures

<table>
<thead>
<tr>
<th>Wed 23</th>
<th>Hashing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fri. March 25</td>
<td>Graphs 1</td>
</tr>
</tbody>
</table>

### Assessments

Assignment 4 will be posted Wednesday, hopefully.