CSE396 Outline Notes
(in process)

Kenneth W. Regan
University at Buffalo (SUNY)

January 29, 2015
Some Larger Questions

1. How do Numbers relate to Strings?
Some Larger Questions

1. How do Numbers relate to Strings?

   - $197 + 48 = 245$. 

Needs two “carries” in decimal. None in binary notation:

$11000101 + 00110000 = 11110101 = 255$.
Some Larger Questions

1. How do Numbers relate to Strings?
   - \( 197 + 48 = 245 \).
   - Needs two “carries” in decimal.
Some Larger Questions

1. How do Numbers relate to Strings?

- $197 + 48 = 245$.
- Needs two “carries” in decimal.
- None in binary notation:

$$11000101 + 00110000 = 11110101 = 255 - 10.$$
Some Larger Questions

1. How do **Numbers** relate to **Strings**?

   - \(197 + 48 = 245\).
   - Needs two “carries” in decimal.
   - None in binary notation:
     \[
     11000101 + 00110000 = 11110101 = 255 - 10.
     \]

   - We use **algorithms** to deal even with basic math.
Some Larger Questions

1. How do Numbers relate to Strings?

- 197 + 48 = 245.
- Needs two “carries” in decimal.
- None in binary notation:

\[
11000101 + 00110000 = 11110101 = 255 - 10.
\]

- We use **algorithms** to deal even with basic math.
- This hints that there’s a lower-level reality.
Some Larger Questions

2. How does an object relate to representations of it?
2. How does an object relate to representations of it?

- “This is Not a Pipe.”
2. How does an **object** relate to **representations** of it?

- “This is Not a Pipe.”
- echo “But is the char after this clause a pipe?” | head
2. How does an object relate to representations of it?

- “This is Not a Pipe.”
- echo “But is the char after this clause a pipe?” | head
- “Syntax Versus Semantics”
Some Larger Questions—3

3. Sets and Logic
3. Sets and Logic
4. What happens when we repeat an operation?
3. Sets and Logic
4. What happens when we repeat an operation?
5. Why does grammar matter when we speak we?
Some Larger Questions—3

3. Sets and Logic
4. What happens when we repeat an operation?
5. Why does grammar matter when we speak we?
6. Can we forecast when a program or process is going to halt?
Strings and Numbers—covered last time.

Sets—were covered last time as "sets of anything." Now we become more specific when building up compound objects.

Compound builders in programming languages:
array, list, struct/record, set, map

Sequences can be infinite, but lists are usually finite, and tuples are always finite.

Many programming languages treat arrays and lists as similar—so will we.

Lists are of the same type, but tuples can have components of different types.

So tuples really model structs/records...

like instance objects of classes.

A 2-tuple is a pair; a 3-tuple is a triple, etc.
Strings and Numbers—covered last time.
Thu. 1/29: Formal Objects and Their Types

- Strings and Numbers—covered last time.
- Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.

Compound builders in programming languages:
- array,
- list,
- struct/record,
- set,
- map

Sequences can be infinite, but lists are usually finite, and tuples are always finite.

Many programming languages treat arrays and lists as similar—so will we.

Lists are of the same type, but tuples can have components of different types.

So tuples really model structs/records...

like instance objects of classes.

A 2-tuple is a pair; a 3-tuple is a triple, etc.
Strings and Numbers—covered last time.

Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.

Compound builders in programming languages: array, list, struct/record, set, map...
Thu. 1/29: Formal Objects and Their Types

- Strings and Numbers—covered last time.
- Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.
- Compound builders in programming languages: array, list, struct/record, set, map...
- *Sequences* can be infinite, but *lists* are usually finite, and *tuples* are always finite.
Strings and Numbers—covered last time.

Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.

Compound builders in programming languages: array, list, struct/record, set, map...

Sequences can be infinite, but lists are usually finite, and tuples are always finite.

Many programming languages treat arrays and lists as similar—so will we.
Strings and Numbers—covered last time.

Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.

Compound builders in programming languages: array, list, struct/record, set, map...

Sequences can be infinite, but lists are usually finite, and tuples are always finite.

Many programming languages treat arrays and lists as similar—so will we.

Lists are of the same type, but tuples can have components of different types.
Strings and Numbers—covered last time.

Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.

Compound builders in programming languages: array, list, struct/record, set, map...

Sequences can be infinite, but lists are usually finite, and tuples are always finite.

Many programming languages treat arrays and lists as similar—so will we.

Lists are of the same type, but tuples can have components of different types.

So tuples really model structs/records...
Strings and Numbers—covered last time.
Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.

Compound builders in programming languages: array, list, struct/record, set, map...

Sequences can be infinite, but lists are usually finite, and tuples are always finite.

Many programming languages treat arrays and lists as similar—so will we.

Lists are of the same type, but tuples can have components of different types.
So tuples really model structs/records...like instance objects of classes.
Strings and Numbers—covered last time.

Sets—were covered last time as “sets of anything.” Now we become more specific when building up compound objects.

Compound builders in programming languages: array, list, struct/record, set, map...

Sequences can be infinite, but lists are usually finite, and tuples are always finite.

Many programming languages treat arrays and lists as similar—so will we.

Lists are of the same type, but tuples can have components of different types.

So tuples really model structs/records...like instance objects of classes.

A 2-tuple is a pair; a 3-tuple is a triple, etc.
Building up the ToC World
Building up the ToC World

- An *alphabet* is a set of characters, presumably finite: Alphabet = set<char>

Strings and numbers are our basic objects, and will sometimes be interchangeable.

A *language* is a set of strings or numbers—usually infinite! Language = set<string> \approx set<int>

Common convention: Lowercase Roman m, n, i, j, k, ... for integer numbers, a, b, c, d, ... for other numbers or chars, x, y, z, w, v, u, ... for strings, uppercase Roman L, A, B, C, D, ... for languages.
Building up the ToC World

- An *alphabet* is a set of characters, presumably finite: \( \text{Alphabet} = \text{set}\langle \text{char}\rangle \)

- A *string* is a *list* of characters over an alphabet: \( \text{string} = \text{list}\langle \text{char}\rangle \)
Building up the ToC World

- An \textit{alphabet} is a set of characters, presumably finite: \( \text{Alphabet} = \text{set<char>} \)
- A \textit{string} is a \textit{list} of characters over an alphabet: \( \text{string} = \text{list<char>} \)
- Strings and numbers are our \textit{basic objects}, and will sometimes be interchangeable.
Building up the ToC World

- An *alphabet* is a set of characters, presumably finite: Alphabet = set<char>
- A *string* is a *list* of characters over an alphabet: string = list<char>
- Strings and numbers are our *basic objects*, and will sometimes be interchangeable.
- A *language* is a set of strings or numbers—usually infinite! Language = set<string> ≈ set<int>
Building up the ToC World

- An *alphabet* is a set of characters, presumably finite: \( \text{Alphabet} = \text{set}\langle \text{char}\rangle \)
- A *string* is a *list* of characters over an alphabet: \( \text{string} = \text{list}\langle \text{char}\rangle \)
- Strings and numbers are our *basic objects*, and will sometimes be interchangeable.
- A *language* is a set of strings or numbers—usually infinite! \( \text{Language} = \text{set}\langle \text{string}\rangle \approx \text{set}\langle \text{int}\rangle \)
- **Common convention**: Lowercase Roman \( m,n,i,j,k,... \) for integer numbers, \( a,b,c,d,... \) for other numbers or chars, \( x,y,z,w,v,u,... \) for strings, uppercase Roman \( L,A,B,C,D,... \) for languages.