

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages**
syntax and semantics
Backus-Naur Form
- 2 Strings, Lists, and Tuples**
composite data types
building data structures
the % operator
- 3 Shared References**
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments**

MCS 260 Lecture 6
Introduction to Computer Science
Jan Verschelde, 25 January 2010

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages
syntax and semantics
Backus-Naur Form
- 2 Strings, Lists, and Tuples
composite data types
building data structures
the % operator
- 3 Shared References
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments

Programming Languages

classified into 4 generations

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 machine language
instructions are encoded as bit sequences
- 2 assembly language
a mnemonic system for representing programs
- 3 high level programming languages
similar to writing algorithms in pseudo code
early examples: FORTRAN, Cobol, C
- 4 problem solving environments
environment helps in discovery of algorithms
mostly limited to one specific problem area

Programming Languages

classified into 4 generations

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 machine language
instructions are encoded as bit sequences
- 2 assembly language
a mnemonic system for representing programs
- 3 high level programming languages
similar to writing algorithms in pseudo code
early examples: FORTRAN, Cobol, C
- 4 problem solving environments
environment helps in discovery of algorithms
mostly limited to one specific problem area

Programming Languages

classified into 4 generations

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 machine language
instructions are encoded as bit sequences
- 2 assembly language
a mnemonic system for representing programs
- 3 high level programming languages
similar to writing algorithms in pseudo code
early examples: FORTRAN, Cobol, C
- 4 problem solving environments
environment helps in discovery of algorithms
mostly limited to one specific problem area

Formal Languages

syntax and semantics

Every language is defined through syntax and semantics.

- Syntax states rules how to compose valid sentences.
- Semantics define the meaning of the sentences.

Syntax and semantics of the assignment:

syntax an assignment consists of a variable,
followed by =, followed by an expression.

semantics the assignment stores the value of the
expression at the right of = into the object
referred to by the variable at the left.

Programming languages have a *context-free grammar*.

Formal Languages

syntax and semantics

Every language is defined through syntax and semantics.

- Syntax states rules how to compose valid sentences.
- Semantics define the meaning of the sentences.

Syntax and semantics of the assignment:

syntax an assignment consists of a variable,
followed by =, followed by an expression.

semantics the assignment stores the value of the
expression at the right of = into the object
referred to by the variable at the left.

Programming languages have a *context-free grammar*.

Formal Languages

syntax and semantics

Every language is defined through syntax and semantics.

- Syntax states rules how to compose valid sentences.
- Semantics define the meaning of the sentences.

Syntax and semantics of the assignment:

syntax an assignment consists of a variable, followed by =, followed by an expression.

semantics the assignment stores the value of the expression at the right of = into the object referred to by the variable at the left.

Programming languages have a *context-free grammar*.

Formal Languages

syntax and semantics

Every language is defined through syntax and semantics.

- Syntax states rules how to compose valid sentences.
- Semantics define the meaning of the sentences.

Syntax and semantics of the assignment:

syntax an assignment consists of a variable, followed by =, followed by an expression.

semantics the assignment stores the value of the expression at the right of = into the object referred to by the variable at the left.

Programming languages have a *context-free grammar*.

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types
building data structures
the % operator

Shared References

a name refers to an object
side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages
syntax and semantics
Backus-Naur Form
- 2 Strings, Lists, and Tuples
composite data types
building data structures
the % operator
- 3 Shared References
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments

Backus-Naur Form

named after John Backus and Peter Naur

The Backus-Naur Form (or BNF) is a formal notation to define the syntax of programming languages.

The BNF of a language consists of

an alphabet is a finite set of symbols, containing terminal and nonterminal symbols.

Keywords are special terminal symbols.

rules of type $A ::= \alpha$ where A is nonterminal, $::=$ is a reserved symbol (in BNF) and α is a string of terminal and nonterminal symbols.

the axiom is the initial symbol.

Sentences are derived by starting with the axiom.

Then we apply the rules, replacing the axiom by strings until the final string only consists of terminal symbols.

Backus-Naur Form

named after John Backus and Peter Naur

The Backus-Naur Form (or BNF) is a formal notation to define the syntax of programming languages.

The BNF of a language consists of

an alphabet is a finite set of symbols, containing terminal and nonterminal symbols.

Keywords are special terminal symbols.

rules of type $A ::= \alpha$ where A is nonterminal, $::=$ is a reserved symbol (in BNF) and α is a string of terminal and nonterminal symbols.

the axiom is the initial symbol.

Sentences are derived by starting with the axiom.

Then we apply the rules, replacing the axiom by strings until the final string only consists of terminal symbols.

Backus-Naur Form

named after John Backus and Peter Naur

The Backus-Naur Form (or BNF) is a formal notation to define the syntax of programming languages.

The BNF of a language consists of

an alphabet is a finite set of symbols, containing terminal and nonterminal symbols.

Keywords are special terminal symbols.

rules of type $A ::= \alpha$ where A is nonterminal, $::=$ is a reserved symbol (in BNF) and α is a string of terminal and nonterminal symbols.

the axiom is the initial symbol.

Sentences are derived by starting with the axiom.

Then we apply the rules, replacing the axiom by strings until the final string only consists of terminal symbols.

Backus-Naur Form

named after John Backus and Peter Naur

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

The Backus-Naur Form (or BNF) is a formal notation to define the syntax of programming languages.

The BNF of a language consists of

an alphabet is a finite set of symbols, containing terminal and nonterminal symbols.

Keywords are special terminal symbols.

rules of type $A ::= \alpha$ where A is nonterminal, $::=$ is a reserved symbol (in BNF) and α is a string of terminal and nonterminal symbols.

the axiom is the initial symbol.

Sentences are derived by starting with the axiom.

Then we apply the rules, replacing the axiom by strings until the final string only consists of terminal symbols.

BNF of a Sum

Let us define the syntax of a sum of natural numbers.

Denote the axiom by S (initial or start symbol).

Nonterminal symbols are enclosed by \langle and \rangle .

Vertical bars $|$ indicate choice, they mean *or*.

$$S ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

$$\langle \text{number} \rangle ::= \langle \text{digit} \rangle | \langle \text{digit} \rangle \langle \text{number} \rangle$$

$$\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$

$$\langle \text{sum} \rangle ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

BNF of a Sum

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Let us define the syntax of a sum of natural numbers.

Denote the axiom by S (initial or start symbol).

Nonterminal symbols are enclosed by \langle and \rangle .

Vertical bars $|$ indicate choice, they mean *or*.

$$S ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

$$\langle \text{number} \rangle ::= \langle \text{digit} \rangle | \langle \text{digit} \rangle \langle \text{number} \rangle$$

$$\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$

$$\langle \text{sum} \rangle ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

BNF of a Sum

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Let us define the syntax of a sum of natural numbers.

Denote the axiom by S (initial or start symbol).

Nonterminal symbols are enclosed by \langle and \rangle .

Vertical bars $|$ indicate choice, they mean *or*.

$$S ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

$$\langle \text{number} \rangle ::= \langle \text{digit} \rangle | \langle \text{digit} \rangle \langle \text{number} \rangle$$

$$\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$

$$\langle \text{sum} \rangle ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

BNF of a Sum

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Let us define the syntax of a sum of natural numbers.

Denote the axiom by S (initial or start symbol).

Nonterminal symbols are enclosed by \langle and \rangle .

Vertical bars $|$ indicate choice, they mean *or*.

$$S ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

$$\langle \text{number} \rangle ::= \langle \text{digit} \rangle | \langle \text{digit} \rangle \langle \text{number} \rangle$$

$$\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$

$$\langle \text{sum} \rangle ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

BNF of a Sum

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Let us define the syntax of a sum of natural numbers.

Denote the axiom by S (initial or start symbol).

Nonterminal symbols are enclosed by \langle and \rangle .

Vertical bars $|$ indicate choice, they mean *or*.

$$S ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

$$\langle \text{number} \rangle ::= \langle \text{digit} \rangle | \langle \text{digit} \rangle \langle \text{number} \rangle$$

$$\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$

$$\langle \text{sum} \rangle ::= \langle \text{number} \rangle | \langle \text{number} \rangle + \langle \text{sum} \rangle$$

Syntax Diagrams

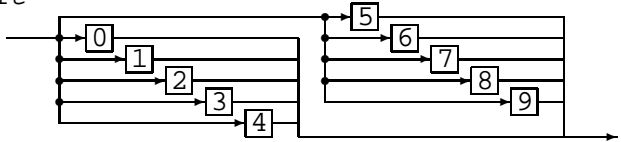
Formal Languages
 syntax and semantics
 Backus-Naur Form

Strings, Lists, and Tuples
 composite data types
 building data structures
 the % operator

Shared References
 a name refers to an object
 side effect of assigning composite objects

Summary + Assignments

digit



number



sum



Syntax Diagrams

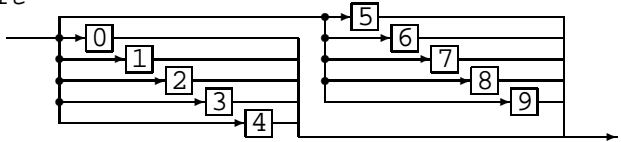
Formal Languages
 syntax and semantics
 Backus-Naur Form

Strings, Lists, and Tuples
 composite data types
 building data structures
 the % operator

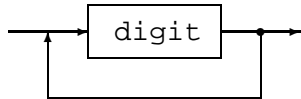
Shared References
 a name refers to an object
 side effect of assigning composite objects

Summary + Assignments

digit



number



sum



Syntax Diagrams

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared

References

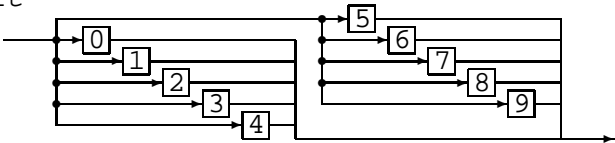
a name refers to an object

side effect of assigning composite objects

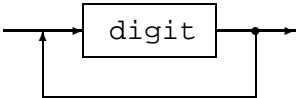
Summary +

Assignments

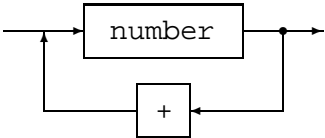
digit



number



sum



Syntax Errors

Interpreters and compilers always first *parse* the statements and check for syntactical correctness.

A *syntax error* means that the statement does not belong to the language.

In the second phase, for valid statements, the interpreter or compiler checks the semantics.

```
>>> float(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Syntax error?

```
>>> flot(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'flot' is not defined
```

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

Syntax Errors

Interpreters and compilers always first *parse* the statements and check for syntactical correctness.

A *syntax error* means that the statement does not belong to the language.

In the second phase, for valid statements, the interpreter or compiler checks the semantics.

```
>>> float(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Syntax error?

```
>>> flot(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'flot' is not defined
```

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

Syntax Errors

Interpreters and compilers always first *parse* the statements and check for syntactical correctness.

A *syntax error* means that the statement does not belong to the language.

In the second phase, for valid statements, the interpreter or compiler checks the semantics.

```
>>> float(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Syntax error?

```
>>> flot(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'flot' is not defined
```

Syntax Errors

Interpreters and compilers always first *parse* the statements and check for syntactical correctness.

A *syntax error* means that the statement does not belong to the language.

In the second phase, for valid statements, the interpreter or compiler checks the semantics.

```
>>> float(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Syntax error?

```
>>> flot(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'flot' is not defined
```

Syntax Errors

Interpreters and compilers always first *parse* the statements and check for syntactical correctness.

A *syntax error* means that the statement does not belong to the language.

In the second phase, for valid statements, the interpreter or compiler checks the semantics.

```
>>> float(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Syntax error?

```
>>> flot(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'flot' is not defined
```

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages
syntax and semantics
Backus-Naur Form
- 2 **Strings, Lists, and Tuples**
composite data types
building data structures
the % operator
- 3 Shared References
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments

Strings as Objects

assigning string constants to variables

String constants are sequences of characters,
enclosed between right quotes.

```
>>> x = 'hello'
>>> x
'hello'
>>> id(x)
406912
>>> x = 'I wrote \"hello\"'
>>> print x
I wrote "hello"
>>> id(x)
421920
```

- different addresses mean different objects
- use `\` to include special characters

Strings as Objects

assigning string constants to variables

String constants are sequences of characters, enclosed between right quotes.

```
>>> x = 'hello'
>>> x
'hello'
>>> id(x)
406912
>>> x = 'I wrote \"hello\"'
>>> print x
I wrote "hello"
>>> id(x)
421920
```

- different addresses mean different objects
- use `\` to include special characters

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Computing with Strings

string operations

With + and * we can calculate with strings:

- + is concatenation
- * is duplication.

```
>>> w = 'we have '  
>>> f = 'fun '  
>>> w + 3*f  
'we have fun fun fun '
```

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Computing with Strings

string operations

With + and * we can calculate with strings:

- + is concatenation
- * is duplication.

```
>>> w = 'we have '  
>>> f = 'fun '  
>>> w + 3*f  
'we have fun fun fun '
```

Manipulating Strings

selecting from strings

Selection: taking slices of strings

```
>>> w = 'hello world'
```

```
>>> len(w)
```

```
11
```

```
>>> w[4]
```

```
'o'
```

```
>>> w[0:5]
```

```
'hello'
```

```
>>> w[-5:]
```

```
'world'
```

- negative indices to count backwards
- an omitted index means first or last

Manipulating Strings

selecting from strings

Selection: taking slices of strings

```
>>> w = 'hello world'
>>> len(w)
11
>>> w[4]
'o'
>>> w[0:5]
'hello'
>>> w[-5:]
'world'
```

- negative indices to count backwards
- an omitted index means first or last

Lists

compounding data

Lists are a versatile compound data type:

- elements in a list can be of different type
- the length of a list is variable.

```
>>> L = [ 'Brian', 4 ]
>>> K = [ 'life' , 'of' ]
>>> K+L
[ 'life', 'of', 'Brian', 4 ]
```

deleting an element for a list:

```
>>> N = _
>>> del(N[3])
>>> N
[ 'life', 'of', 'Brian' ]
```

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Lists are a versatile compound data type:

- elements in a list can be of different type
- the length of a list is variable.

```
>>> L = [ 'Brian' , 4 ]
>>> K = [ 'life' , 'of' ]
>>> K+L
[ 'life' , 'of' , 'Brian' , 4 ]
```

deleting an element for a list:

```
>>> N = _
>>> del(N[3])
>>> N
[ 'life' , 'of' , 'Brian' ]
```

Lists are a versatile compound data type:

- elements in a list can be of different type
- the length of a list is variable.

```
>>> L = [ 'Brian' , 4 ]
>>> K = [ 'life' , 'of' ]
>>> K+L
[ 'life' , 'of' , 'Brian' , 4 ]
```

deleting an element for a list:

```
>>> N = _
>>> del(N[3])
>>> N
[ 'life' , 'of' , 'Brian' ]
```

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Reversing the characters in a string:

```
>>> s = "monday"
>>> L = [a for a in s]
>>> L
['m', 'o', 'n', 'd', 'a', 'y']
>>> L.reverse()
>>> L
['y', 'a', 'd', 'n', 'o', 'm']
>>> t = ''.join(L)
>>> t
'yadnom'
```

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Reversing the characters in a string:

```
>>> s = "monday"
>>> L = [a for a in s]
>>> L
['m', 'o', 'n', 'd', 'a', 'y']
>>> L.reverse()
>>> L
['y', 'a', 'd', 'n', 'o', 'm']
>>> t = ''.join(L)
>>> t
'yadnom'
```

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Reversing the characters in a string:

```
>>> s = "monday"
>>> L = [a for a in s]
>>> L
['m', 'o', 'n', 'd', 'a', 'y']
>>> L.reverse()
>>> L
['y', 'a', 'd', 'n', 'o', 'm']
>>> t = ''.join(L)
>>> t
'yadnom'
```

Tuples

lists of fixed length

Tuples are very similar to lists,
except that their length remains fixed.

Good for storing coordinates of points or vectors.

```
>>> A = (1, 2, 3)
```

```
>>> A
(1, 2, 3)
```

Notice the round brackets. We can make longer tuples:

```
>>> B = (A[0], A[1], A[2], 'go')
```

```
>>> B
(1, 2, 3, 'go')
```

Notice: `A[:]` = `(A[0], A[1], A[2])`.

Tuples

lists of fixed length

Tuples are very similar to lists,
except that their length remains fixed.
Good for storing coordinates of points or vectors.

```
>>> A = (1, 2, 3)
```

```
>>> A  
(1, 2, 3)
```

Notice the round brackets. We can make longer tuples:

```
>>> B = (A[0], A[1], A[2], 'go')
```

```
>>> B  
(1, 2, 3, 'go')
```

Notice: `A[:]` = `(A[0], A[1], A[2])`.

Tuples

lists of fixed length

Tuples are very similar to lists,
except that their length remains fixed.
Good for storing coordinates of points or vectors.

```
>>> A = (1, 2, 3)
```

```
>>> A  
(1, 2, 3)
```

Notice the round brackets. We can make longer tuples:

```
>>> B = (A[0], A[1], A[2], 'go')
```

```
>>> B  
(1, 2, 3, 'go')
```

Notice: `A[:]` = `(A[0], A[1], A[2])`.

the `in` and `len` operators

We define a string `s`, tuple `t`, and list `L`:

```
>>> s = "hello"  
>>> t = ('h', 'e', 'l', 'l', 'o')  
>>> L = ['h', 'e', 'l', 'l', 'o']
```

The `in` method applies to all three types:

```
>>> 'e' in s  
True  
>>> 'e' in t  
True  
>>> 'e' in L  
True
```

Similarly: `len(s)`, `len(t)`, and `len(L)`
return the length of a string `s`, tuple `t`, and list `L`.

the `in` and `len` operators

We define a string `s`, tuple `t`, and list `L`:

```
>>> s = "hello"
>>> t = ('h','e','l','l','o')
>>> L = ['h','e','l','l','o']
```

The `in` method applies to all three types:

```
>>> 'e' in s
True
>>> 'e' in t
True
>>> 'e' in L
True
```

Similarly: `len(s)`, `len(t)`, and `len(L)`
return the length of a string `s`, tuple `t`, and list `L`.

the `in` and `len` operators

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

We define a string `s`, tuple `t`, and list `L`:

```
>>> s = "hello"
>>> t = ('h','e','l','l','o')
>>> L = ['h','e','l','l','o']
```

The `in` method applies to all three types:

```
>>> 'e' in s
True
>>> 'e' in t
True
>>> 'e' in L
True
```

Similarly: `len(s)`, `len(t)`, and `len(L)` return the length of a string `s`, tuple `t`, and list `L`.

input() for Composite Types

not restricted to numbers

The user may also give tuples and lists:

```
>>> p = input('give coordinates : ')
give coordinates : (1,3.4,9)
```

```
>>> p
(1, 3.3999999999999999, 9)
```

```
>>> type(p)
<type 'tuple'>
```

```
>>> L = input('list fruit : ')
list food : [ 'apple' , 'orange' ]
```

```
>>> L
['apple', 'orange']
```

```
>>> type(L)
<type 'list'>
```

Use upper case L instead of l to avoid confusion with 1.

input() for Composite Types

not restricted to numbers

The user may also give tuples and lists:

```
>>> p = input('give coordinates : ')
give coordinates : (1,3.4,9)
```

```
>>> p
(1, 3.3999999999999999, 9)
```

```
>>> type(p)
<type 'tuple'>
```

```
>>> L = input('list fruit : ')
list food : [ 'apple' , 'orange' ]
```

```
>>> L
['apple', 'orange']
>>> type(L)
<type 'list'>
```

Use upper case L instead of l to avoid confusion with 1.

functions `str()` and `eval()`

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

```
>>> L = range(0,4)
>>> L
[0, 1, 2, 3]
>>> s = str(L)
>>> s
'[0, 1, 2, 3]'
```

Via `str()` we convert an object to its string representation.

To revert:

```
>>> K = eval(s)
>>> K
[0, 1, 2, 3]
```

The `eval()` evaluates a string, calling the Python interpreter on the expression.

functions `str()` and `eval()`

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

```
>>> L = range(0,4)
>>> L
[0, 1, 2, 3]
>>> s = str(L)
>>> s
'[0, 1, 2, 3]'
```

Via `str()` we convert an object to its string representation.
To revert:

```
>>> K = eval(s)
>>> K
[0, 1, 2, 3]
```

The `eval()` evaluates a string, calling the Python interpreter on the expression.

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages
syntax and semantics
Backus-Naur Form
- 2 **Strings, Lists, and Tuples**
composite data types
building data structures
the % operator
- 3 Shared References
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments

Building Data Structures

nesting tuples

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

The matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ can be stored as

```
>>> m = ((1, 2), (3, 4))
```

```
>>> m
```

```
((1, 2), (3, 4))
```

```
>>> m[0]
```

```
(1, 2)
```

```
>>> m[1]
```

```
(3, 4)
```

```
>>> m[0][1]
```

```
2
```

Building Data Structures

nesting lists

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

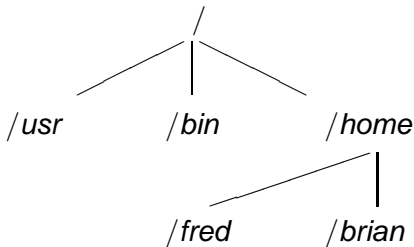
Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

To store a directory tree



we could use the following nested list:

```

>>> L = ['/usr', '/bin', ['/home/fred', '/home/brian']]
>>> L
['/usr', '/bin', ['/home/fred', '/home/brian']]
  
```

Building Data Structures

nesting lists

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

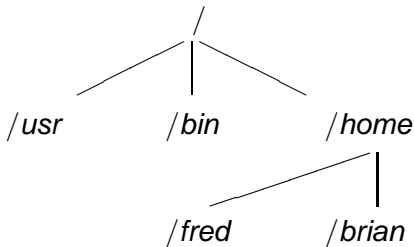
Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

To store a directory tree



we could use the following nested list:

```

>>> L = ['/usr', '/bin', ['/home/fred', '/home/brian']]
>>> L
['/usr', '/bin', ['/home/fred', '/home/brian']]
  
```

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages
syntax and semantics
Backus-Naur Form
- 2 **Strings, Lists, and Tuples**
composite data types
building data structures
the % operator
- 3 Shared References
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments

The % Operator

to format output

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

% turns objects to strings for formatted output

% takes multiple arguments to format

The general syntax is

```
print_string % (convert_arguments)
```

- `print_string` contains the string to be printed as it is, with % codes to format objects
- `convert_arguments` contain the data to be converted, either one single item, tuple or dictionary.

```
>>> s = 'The grade of %s is %d.' % ('Brian',7)
>>> s
'The grade of Brian is 7.'
```

The % Operator

to format output

- turns objects to strings for formatted output
- takes multiple arguments to format

The general syntax is

```
print_string % (convert_arguments)
```

- `print_string` contains the string to be printed as it is, with % codes to format objects
- `convert_arguments` contain the data to be converted, either one single item, tuple or dictionary.

```
>>> s = 'The grade of %s is %d.' % ('Brian',7)
>>> s
'The grade of Brian is 7.'
```

The % Operator

to format output

- turns objects to strings for formatted output
- takes multiple arguments to format

The general syntax is

```
print_string % (convert_arguments)
```

- `print_string` contains the string to be printed as it is, with % codes to format objects
- `convert_arguments` contain the data to be converted, either one single item, tuple or dictionary.

```
>>> s = 'The grade of %s is %d.' % ('Brian',7)
>>> s
'The grade of Brian is 7.'
```

% Codes for Strings

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

| | |
|-----------------|--|
| <code>%c</code> | converts to character, see ASCII code |
| <code>%s</code> | converts to string, applies <code>str()</code> |

Every symbol has an ASCII code:

```
>>> ord(';')
59
>>> '%c' % 59
';'
```

Turning numbers to strings:

```
>>> str(343.23)
'343.23'
>>> '%20s' % 343.23
'                343.23'
```

Try also `'%-20s'`.

% Codes for Strings

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

| | |
|-----------------|--|
| <code>%c</code> | converts to character, see ASCII code |
| <code>%s</code> | converts to string, applies <code>str()</code> |

Every symbol has an ASCII code:

```
>>> ord(';')
59
>>> '%c' % 59
';'
```

Turning numbers to strings:

```
>>> str(343.23)
'343.23'
>>> '%20s' % 343.23
'                343.23'
```

Try also `'%-20s'`.

% Codes for Strings

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

| | |
|-----------------|--|
| <code>%c</code> | converts to character, see ASCII code |
| <code>%s</code> | converts to string, applies <code>str()</code> |

Every symbol has an ASCII code:

```
>>> ord(';')
59
>>> '%c' % 59
';'
```

Turning numbers to strings:

```
>>> str(343.23)
'343.23'
>>> '%20s' % 343.23
'                343.23'
```

Try also `'%-20s'`.

Formatting Integers

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

| code | converts to ... |
|------|--|
| %i | a signed decimal integer |
| %d | a signed decimal integer |
| %u | an unsigned decimal integer |
| %o | an octal integer |
| %x | an hexadecimal integer, lower case letters |
| %X | an hexadecimal integer, upper case letters |

Always display the sign:

```
>>> '%+d' % -23
```

```
'-23'
```

```
>>> '%+d' % 23
```

```
'+23'
```

Notice: `'+%d' ≠ '%+d'!`

Hexadecimals are shorthand notations for binary:

```
>>> '%X' % 2007
```

```
'7D7'
```

How many bits?

Formatting Integers

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

| code | converts to ... |
|------|--|
| %i | a signed decimal integer |
| %d | a signed decimal integer |
| %u | an unsigned decimal integer |
| %o | an octal integer |
| %x | an hexadecimal integer, lower case letters |
| %X | an hexadecimal integer, upper case letters |

Always display the sign:

```
>>> '%+d' % -23
```

```
'-23'
```

```
>>> '%+d' % 23
```

```
'+23'
```

Notice: `'+%d' ≠ '%+d'!`

Hexadecimals are shorthand notations for binary:

```
>>> '%X' % 2007
```

```
'7D7'
```

How many bits?

Formatting Integers

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

| code | converts to ... |
|------|--|
| %i | a signed decimal integer |
| %d | a signed decimal integer |
| %u | an unsigned decimal integer |
| %o | an octal integer |
| %x | an hexadecimal integer, lower case letters |
| %X | an hexadecimal integer, upper case letters |

Always display the sign:

```
>>> '%+d' % -23
```

```
'-23'
```

```
>>> '%+d' % 23
```

```
'+23'
```

Notice: `'+%d' ≠ '%+d'!`

Hexadecimals are shorthand notations for binary:

```
>>> '%X' % 2007
```

```
'7D7'
```

How many bits?

Formatting Floats

shortening decimals and scientific notation

| code | converts to ... |
|-----------------|--|
| <code>%f</code> | a floating-point real number |
| <code>%e</code> | scientific notation, using <code>e</code> |
| <code>%E</code> | scientific notation, using <code>E</code> |
| <code>%g</code> | the value shorter of <code>%f</code> and <code>%e</code> |
| <code>%G</code> | the value shorter of <code>%f</code> and <code>%E</code> |

Some examples:

```
>>> the_e = math.exp(1)
>>> '%+8.4f' % the_e
' +2.7183'
>>> '%+12.4E' % the_e
' +2.7183E+00'
>>> '%+12.4g' % the_e
'          +2.718'
```

Raw Strings

strings with special characters

With `\n` we begin a new line:

```
>>> print 'hello\nworld'  
hello  
world
```

Suppose we want to print `\n` as well:

```
>>> print r'hello\nworld'  
hello\nworld
```

The `r` before a string converts it into a *raw* string.

A *raw string* may contain special characters.

In a regular string, these special characters are converted into escape characters.

Raw Strings

strings with special characters

With `\n` we begin a new line:

```
>>> print 'hello\nworld'  
hello  
world
```

Suppose we want to print `\n` as well:

```
>>> print r'hello\nworld'  
hello\nworld
```

The **r** before a string converts it into a *raw* string.

A *raw string* may contain special characters.

In a regular string, these special characters are converted into escape characters.

Raw Strings

strings with special characters

With `\n` we begin a new line:

```
>>> print 'hello\nworld'  
hello  
world
```

Suppose we want to print `\n` as well:

```
>>> print r'hello\nworld'  
hello\nworld
```

The **r** before a string converts it into a *raw* string.

A *raw string* may contain special characters.

In a regular string, these special characters are converted into escape characters.

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages
syntax and semantics
Backus-Naur Form
- 2 Strings, Lists, and Tuples
composite data types
building data structures
the % operator
- 3 Shared References
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
References**a name refers to an
object**side effect of
assigning composite
objectsSummary +
Assignments

Names are References

variable names refer to objects

```
>>> s = 'a'
```

```
>>> t = s
```

The variables s and t refer both to the same object.

```
>>> s = 'b'
```

```
>>> t
```

```
'a'
```

```
>>> s
```

```
'b'
```

Now s and t refer to different objects.

Although strings are sequences of characters, in Python strings are basic data types, like numbers, unlike lists, tuples, dictionaries.

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Names are References

variable names refer to objects

```
>>> s = 'a'
```

```
>>> t = s
```

The variables s and t refer both to the same object.

```
>>> s = 'b'
```

```
>>> t
```

```
'a'
```

```
>>> s
```

```
'b'
```

Now s and t refer to different objects.

Although strings are sequences of characters, in Python strings are basic data types, like numbers, unlike lists, tuples, dictionaries.

Formal
Languagessyntax and
semantics

Backus-Naur Form

Strings, Lists,
and Tuples

composite data types

building data
structures

the % operator

Shared
Referencesa name refers to an
objectside effect of
assigning composite
objectsSummary +
Assignments

Names are References

variable names refer to objects

```
>>> s = 'a'
```

```
>>> t = s
```

The variables s and t refer both to the same object.

```
>>> s = 'b'
```

```
>>> t
```

```
'a'
```

```
>>> s
```

```
'b'
```

Now s and t refer to different objects.

Although strings are sequences of characters, in Python strings are basic data types, like numbers, unlike lists, tuples, dictionaries.

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

- 1 Formal Languages
syntax and semantics
Backus-Naur Form
- 2 Strings, Lists, and Tuples
composite data types
building data structures
the % operator
- 3 Shared References
a name refers to an object
side effect of assigning composite objects
- 4 Summary + Assignments

Shared References

side effect of assigning composite objects

Consider lists of strings:

```
>>> L = [ 'a' ]
```

```
>>> K = L
```

Both L and K refer to the same list.

List are composite data types, change a component:

```
>>> L[0] = 'b'
```

```
>>> K
```

```
[ 'b' ]
```

the change in L also changed K!

The same list referred to by both L and K
is composed of items, objects stored only once.

Shared References

side effect of assigning composite objects

Consider lists of strings:

```
>>> L = [ 'a' ]
```

```
>>> K = L
```

Both L and K refer to the same list.

List are composite data types, change a component:

```
>>> L[0] = 'b'
```

```
>>> K
```

```
[ 'b' ]
```

the change in L also changed K!

The same list referred to by both L and K
is composed of items, objects stored only once.

Shared References

side effect of assigning composite objects

Consider lists of strings:

```
>>> L = [ 'a' ]
```

```
>>> K = L
```

Both L and K refer to the same list.

List are composite data types, change a component:

```
>>> L[0] = 'b'
```

```
>>> K
```

```
[ 'b' ]
```

the change in L also changed K!

The same list referred to by both L and K
is composed of items, objects stored only once.

Avoiding Shared References

Sometimes we may *on purpose* put brackets around an object to create a shared reference.

But we can avoid sharing by copying the content:

```
>>> L = [ 'a' ]
>>> K = [L[0]]
>>> K
[ 'a' ]
```

L and K refer to different lists with the same content.

```
>>> L[0] = 'b'
>>> K
[ 'a' ]
>>> L
[ 'b' ]
```

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

Avoiding Shared References

Sometimes we may *on purpose* put brackets around an object to create a shared reference.

But we can avoid sharing by copying the content:

```
>>> L = [ 'a' ]
>>> K = [L[0]]
>>> K
[ 'a' ]
```

L and K refer to different lists with the same content.

```
>>> L[0] = 'b'
>>> K
[ 'a' ]
>>> L
[ 'b' ]
```

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

Avoiding Shared References

Formal Languages

syntax and semantics

Backus-Naur Form

Strings, Lists, and Tuples

composite data types

building data structures

the % operator

Shared References

a name refers to an object

side effect of assigning composite objects

Summary + Assignments

Sometimes we may *on purpose* put brackets around an object to create a shared reference.

But we can avoid sharing by copying the content:

```
>>> L = [ 'a' ]
>>> K = [ L[0] ]
>>> K
[ 'a' ]
```

L and K refer to different lists with the same content.

```
>>> L[0] = 'b'
>>> K
[ 'a' ]
>>> L
[ 'b' ]
```

Summary + Assignments

In this lecture we covered

- section 6.1, 6.4 in *Computer Science. An Overview*
- start of chapters 3 & 4 of *Python Programming*.

Assignments:

- 1 Give the BNF of a floating-point number.
- 2 Extend BNF for a sum to that of a polynomial in one variable, in fully expanded form.
- 3 Find the BNF of the Python language.
- 4 Apply the % operator to print rational numbers given by numerator n and denominator d as n/d . Use it in

```
$ python printrat.py
Give numerator : 4352
Give denominator : 234249
Your number : 4352/234249.
```

- 5 Compare $2^{**}3^{**}4$ with $(2^{**}3)^{**}4$ and explain.