Outline

1. Boolean Algebra
   - logical expressions
   - pseudocode and flowcharts

2. Conditional Constructs
   - conditional operators
   - if, else, elif

3. Logic in Sage
   - computing truth tables with Sage

4. Summary + Assignments
Boolean Algebra, Flowcharts
Conditional Expressions

1. Boolean Algebra
   - logical expressions
   - pseudocode and flowcharts

2. Conditional Constructs
   - conditional operators
   - if, else, elif

3. Logic in Sage
   - computing truth tables with Sage

4. Summary + Assignments
Boolean Algebra
computing with logical expressions

Boolean algebra is the calculation with True and False (often having values 1 and 0). The operators are and, or, and not. Truth tables define the outcome for all values:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x and y</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x or y</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Evaluation Laws
law and order in the Boolean algebra

When not, and, or occur in an expression, not is first evaluated, before and, and finally or.

De Morgan’s laws for simplifying expressions:

- not (( not x ) or ( not y )) = x and y
  Negating not being alive or not being well means being alive and being well.

- not (( not x ) and ( not y )) = x or y
  Negating not going to school and not going to work means going to school or going to work.

We prove these laws by truth tables.
Applications of truth tables:

- Realization of electronic circuits.

- Simplification of conditional statements in programs.
Boolean Algebra, Flowcharts  
Conditional Expressions

1. Boolean Algebra  
   - logical expressions  
   - pseudocode and flowcharts

2. Conditional Constructs  
   - conditional operators  
   - if, else, elif

3. Logic in Sage  
   - computing truth tables with Sage

4. Summary + Assignments
The Absolute Value – an example of an `if` statement

The function `abs` is available in Python:

```python
>>> abs(-3.5)
3.5
>>> abs(3.5)
3.5
```

The mathematical definition of `abs(x)` as $y = |x|$: 

$$
|x| = \begin{cases} 
  x & \text{if } x \geq 0 \\
  -x & \text{if } x < 0
\end{cases}
$$
Pseudocode
to formally describe algorithms

To develop and define an algorithm, we use *pseudocode*. Pseudocode is not real code, but to the reader it has the same properties as a formal language.

Example: print the absolute value of a number.

The number is given by the user.

In words, we could describe the program as:

```
ask the user for a number;
if the number is less than zero,
    then print – before the number;
else print the number.
```

Mix of formal *if*, *then*, and *else* with English.
Printing the absolute value of a number:

```
x = input()

x < 0?
  True
    print -x
  False
    print x
```

Flowcharts schematically represent the logical flow.
Boolean Algebra, Flowcharts

Conditional Expressions

1. Boolean Algebra
   - logical expressions
   - pseudocode and flowcharts

2. Conditional Constructs
   - conditional operators
   - if, else, elif

3. Logic in Sage
   - computing truth tables with Sage

4. Summary + Assignments
Comparison Operators

The outcome of a comparison is True or False:

```python
>>> 1 < 7
True
```  
```python
>>> 1 >= 7
False
```  

The comparison operators:

- `==` : `x == y` is equal?
- `!=` : `x != y` not equal?
- `<` : `x < y` less than?
- `>` : `x > y` greater than?
- `<=` : `x <= y` less or equal?
- `>=` : `x >= y` greater or equal?
The *is* Operator – equal values but different objects

Testing whether composite objects are equal:

```python
>>> L = [2,3]; K = [2,3]
>>> L == K
True
>>> L is K
False
```

*L* and *K* refer to

- lists which contain the **same values**, 
- lists stored as **different objects**.

```python
>>> M = L; M is L
True
>>> M == L
True
```
Boolean Operators

Combining results of logical expressions:

```python
>>> x = 3
>>> (x > 0) and (x < 10)
True
>>> (x < 0) or (x < 5)
True
>>> not (x < 0)
True
```

The brackets are not needed.

```
and x and y  both True?
or x or y    is one True?
not not x    is False?
```
Printing Booleans – as numbers or strings

Type `bool` is another elementary data type:

```python
>>> type(True)
<type 'bool'>
```

Although `True` is 1 and `False` is 0:

```python
>>> '%d' % True
'1'
>>> '%d' % False
'0'
```

Printing booleans as strings:

```python
>>> str(True)
'True'
>>> '%s' % True
'True'
```
Boolean Algebra, Flowcharts

Conditional Expressions

1. Boolean Algebra
   - logical expressions
   - pseudocode and flowcharts

2. Conditional Constructs
   - conditional operators
   - if, else, elif

3. Logic in Sage
   - computing truth tables with Sage

4. Summary + Assignments
The if Statement – conditional execution of code

The syntax of the if:

```python
if < condition >:
    < statements when condition is True >
```

All statements to be executed only if the condition is true must be preceded by the right intendations!

Suppose we want to print the ’+’ for positive numbers.

With an if we could do it as follows:

```python
if x > 0:
    print ’+’
    print x
```

Only the second one works correctly for all x.
the if else statement

The syntax of the *if else*:

```python
if < condition >:
    < statements when condition is True >
else:
    < statements when condition is False >
```

Printing the absolute value of a number:

```python
if x < 0:
    print -x
else:
    print x
```
This program prompts the user to enter a number. If the number entered is larger or equal than 80, then the user is congratulated, else we are sorry.

```
DATA = input('Enter your number : ')
NUMBER = int(DATA)
if NUMBER >= 80:
    print('Congratulations. You passed!')
else:
    print('Sorry. Please try again...')
```
n = input('Enter your number :')

- If $n \geq 90$, then grade = A
- If $n \geq 80$, then grade = B
- If $n \geq 70$, then grade = C
- If $n \geq 60$, then grade = D
- Otherwise, grade = F
if elif else to choose between multiple alternatives

The syntax of the `if elif else`:

```python
if < condition 1 >:
    < statements when condition 1 is True >
elif < condition 2 >:
    < statements when condition 2 is True >
...
eelif < condition n >:
    < statements when condition n is True >
else:
    < statements when everything is False >
```

The conditions are evaluated in the order as they appear.
The grade for a course is represented by a letter. We compute the grade along a scale.

```python
DATA = input('Enter your number : ')
NUMBER = int(DATA)
if NUMBER >= 90:
    GRADE = 'A'
elif NUMBER >= 80:
    GRADE = 'B'
elif NUMBER >= 70:
    GRADE = 'C'
elif NUMBER >= 60:
    GRADE = 'D'
else:
    GRADE = 'F'
print('Your grade is ' + GRADE + '.')
```
Nested if else statements – follow up questions

Statements following if or else can again be conditional. Nested if statements are good for dialogues with a user, when the outcome cannot be anticipated:

```python
ANS = input('happy ? (y/n) ')
if ANS == 'n':
    ANS = input('bored ? (y/n) ')
    if ANS == 'y':
        print('class is soon over')
    else:
        print('but it is Friday')
else:
    print('keep up the good work')
```

Python gives an error when = is used instead of ==.
Boolean Algebra, Flowcharts
Conditional Expressions

1. Boolean Algebra
   - logical expressions
   - pseudocode and flowcharts

2. Conditional Constructs
   - conditional operators
   - if, else, elif

3. Logic in Sage
   - computing truth tables with Sage

4. Summary + Assignments
Truth Tables in Sage – using SymbolicLogic

Run SageMathCell at https://sagecell.sagemath.org:

```sage
sage: logic = SymbolicLogic()
sage: s = logic.statement("a&b")

Instead of `and`, use the `&` operator.

sage: t = logic.truthtable(s)
sage: logic.print_table(t)
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Truth Tables continued – the or operation

Sage session continued...

sage: s = logic.statement("a|b")

Instead of or, use the | operator.

sage: logic.print_table(logic.truthtable(s))

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Truth Tables continued – the not operation

Sage session continued...

```python
sage: s = logic.statement("!a")

Instead of `not`, use the `!` operator.

```python
sage: t = logic.truthtable(s)
sage: logic.print_table(t)

<table>
<thead>
<tr>
<th>a</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
```
Proving De Morgan’s Law – with truth tables

The first law of De Morgan:

\[ \neg ((\neg x) \lor (\neg y)) = x \land y \]

Sage session continued ...

```
sage: law = '!((!x)|(!y))'
sage: s = logic.statement(law)
sage: logic.print_table(t)
x  |  y  |  value  |
------------------------
False | False | False  |
False | True  | False  |
True  | False | False  |
True  | True  | True   |
```

We recognize the truth table for \( x \land y \).
Summary + Assignments

In this lecture we covered

- section 1.1 in *Computer Science: an overview*
- sections 2.6.1 to 2.6.3 of *Python Programming*

Assignments:

1. Omit the brackets in De Morgan’s Laws and create a truth table to evaluate the expressions.

2. Draw a flowchart for the code using a nested if else for the followup questions "happy ?" and "bored ?".

3. Write pseudocode and draw of flowchart for a program that reads in a positive number and prints out whether the number is divisible by 2, 3, 5, or not.

4. Define a Python dictionary for the truth table of $x$ and $y$. 