3. Conditional Execution

Topics:
- Boolean values
- Relational operators
- if statements
- The Boolean type

Motivation

Problem:
Assign positive float values to variables a and b and print the values a**b and b**a.

Solution:
```python
a = input('Enter a pos float: ')
b = input('Enter a pos float: ')
print a**b, b**a
```

Solution Using If-Else

```python
a = input('Enter a pos float: ')
b = input('Enter a pos float: ')
aTob = a**b
bToa = b**a
if aTob > bToa:
    print aTob
else:
    print bToa
```

This is what is called "conditional execution."

If-Else: How Does it Work?

```python
aTob = a**b
bToa = b**a
if aTob > bToa:
    print aTob
else:
    print bToa
```

Let's suppose the value of a is 2 and the value of b is 7.

Is the value of aTob larger than the value of bToa?
Solution Using If-Else

\[
a_{\text{ToB}} = a^b \\
b_{\text{ToA}} = b^a \\
\text{if } a_{\text{ToB}} > b_{\text{ToA}}: \\
\quad \text{print } a_{\text{ToB}} \\
\text{else:} \\
\quad \text{print } b_{\text{ToA}}
\]

Is the value of \(a_{\text{ToB}}\) larger than the value of \(b_{\text{ToA}}\)? Yes!

If-Else: How Does it Work?

\[
a_{\text{ToB}} = a^b \\
b_{\text{ToA}} = b^a \\
\text{if } a_{\text{ToB}} > b_{\text{ToA}}: \\
\quad \text{print } a_{\text{ToB}} \\
\text{else:} \\
\quad \text{print } b_{\text{ToA}}
\]

Now let's suppose the value of \(a\) is 7 and the value of \(b\) is 2.

\[
a_{\text{ToB}} = a^b \\
b_{\text{ToA}} = b^a \\
\text{if } a_{\text{ToB}} > b_{\text{ToA}}: \\
\quad \text{print } a_{\text{ToB}} \\
\text{else:} \\
\quad \text{print } b_{\text{ToA}}
\]

Is the value of \(a_{\text{ToB}}\) larger than the value of \(b_{\text{ToA}}\)? No!

If-Else: How Does it Work?

\[
a_{\text{ToB}} = a^b \\
b_{\text{ToA}} = b^a \\
\text{if } a_{\text{ToB}} > b_{\text{ToA}}: \\
\quad \text{print } a_{\text{ToB}} \\
\text{else:} \\
\quad \text{print } b_{\text{ToA}}
\]

Note the punctuation and the indentation. This is essential syntax. Forgetting the colons is a major boo boo!

“Synonym”

\[
a_{\text{ToB}} = a^b \\
b_{\text{ToA}} = b^a \\
\text{if } a_{\text{ToB}} > b_{\text{ToA}}: \\
\quad \text{print } a_{\text{ToB}} \\
\text{else:} \\
\quad \text{print } b_{\text{ToA}}
\]

In a comparison, legal to have general expressions on either side of the "<".
### The if-else Construction

```python
if Boolean expression:
    Statements to execute if the expression is True
else:
    Statements to execute if the expression is False
```

This is an example of conditional execution. The if-else construction is sometimes called "alternative execution".

### Another Example

#### Problem:

The last character in a string 5-character string is 'y'. Change the 'y' to 'i' and add 'es'.

#### Solution:

```python
s = s[0:4] + 'ies'
```

Want 'carry' to become 'carries'

Use string slicing and concatenation: 'car' + 'ies'

### A Modified Problem

If the last character in a 5-character string `s` is 'y', then

1. change the 'y' to 'i'
2. add 'es'
3. assign the result to a variable `plural`

Otherwise, just add 's' and assign the result to a variable `plural`.

This will require the if-else construction.

```python
if s[4] == 'y':
    plural = s[0:4] + 'ies'
else:
    plural = s + 's'
print s, plural
```

Remember: s[0:4] names the substring comprised of the first 4 characters.
Discussion of Solution

```python
if s[4]=='y':
    plural = s[0:4] + 'ies'
else:
    plural = s + 's'
print s, plural
```

A new comparison is being used.

If you want to check to see if two expressions have the same value, use `==`.


Relational Operators

```plaintext
<  Less than  
>  Greater than  
<= Less than or equal to  
>= Greater than or equal to  
== Equal to  
!= Not equal to
```

Relational Operators in Action

```plaintext
x ---> 3  y ---> 6

x < y True
2*x > y False
x <= y True
x >= y False
x == y/2 True
x != y/2 False
```

Boolean Operations with Strings

Comparing for equality...

```python
>>> s = 'abc'
>>> s == 'abc'
True
>>> s == 'abc '
False
```

Boolean Operations with Strings

Comparing for alphabetical order...

```python
>>> s = 'Dog'
>>> s > 'Horse'
False
>>> s < 'Horse'
True
>>> s < 'dog'
True
```

Two strings are equal if they have the same length and agree in each position.

If the expression on the left is a different numerical type then the expression on the right, everything is converted to float.

Comparing for alphabetical order...
Relational Operators in Action

\[
\begin{align*}
&x \rightarrow \text{'key'} \quad y \rightarrow \text{'hockey'} \\
x < y \quad \text{False} \\
x > y \quad \text{True} \\
'hoc' + x \leq y \quad \text{True} \\
x \geq y \quad \text{True} \\
x = y[3:] \quad \text{True} \\
x \neq x + ' ' \quad \text{True}
\end{align*}
\]

Comparisons based on alphabetical order. 'key' is false because 'key' does not come before 'hockey' in the dictionary.

Another Problem

Assume that s1 and s2 are initialized strings.

Write code that prints them in alphabetical order on separate lines.

Solution

```python
if s1 < s2:
  print s1
  print s2
else:
  print s2
  print s1
```

s1 < s2
Is this True or False?

Solution

```python
if s1 < s2:
  print s1
  print s2
else:
  print s2
  print s1
```

s1 < s2
It's true!
Output:
```
cat
dog
```

Solution

```python
if s1 < s2:
  print s1
  print s2
else:
  print s2
  print s1
```

s1 < s2
Is this True or False?

Solution

```python
if s1 < s2:
  print s1
  print s2
else:
  print s2
  print s1
```

s1 < s2
It's false!
Output:
```
cat
dog
```
Indentation Is Important

```python
if s1 < s2:
    print s1
    print s2
else:
    print s2
    print s1
```

Output:
```
cat
dog
cat
```

What if You Have More than Two Alternatives?

For example, given a numerical test score between 0 and 100, print out the letter grade equivalent according to these rules:

- A: 90-100
- B: 80-89
- C: 70-79
- U: <70

The If-Elif-Else Construction

```python
x = input('Score: ')
if x >= 90:
    grade = 'A'
elif x >= 80:
    grade = 'B'
elif x >= 70:
    grade = 'C'
else:
    grade = 'U'
print grade
```

Read "elif" as "else if"

If-Elif-Else: How it Works

```python
x = input('Score: ')
if x >= 90:
    grade = 'A'
elif x >= 80:
    grade = 'B'
elif x >= 70:
    grade = 'C'
else:
    grade = 'U'
print grade
```

1. Is this true?
   2. No.
   3. Proceed to the next comparison.
If-Elif-Else: How it Works

```python
x = input('Score: ')  
if x>=90:  
    grade = 'A'  
elif x>=80:  
    grade = 'B'  
elif x>=70:  
    grade = 'C'  
else:  
    grade = 'U'  
print grade
```

The indentation scheme "tells" Python what comes after the if-elif-else.

1. Is this true?
2. No.
3. Proceed to the next comparison.

1. Is this true?
2. Yes.
3. Execute the statement(s) it guards and proceed to whatever follows the if-elif-else.

1. Is this true?
2. No.
3. Proceed to the next comparison.

1. Is this true?
2. Yes.
3. Execute the statement(s) it guards and proceed to whatever follows the if-elif-else.

1. Is this true?
2. Yes.
3. Execute the statement(s) it guards and proceed to whatever follows the if-elif-else.

1. Is this true?
2. Yes.
3. Execute the statement(s) it guards and proceed to whatever follows the if-elif-else.

1. Is this true?
2. Yes.
3. Execute the statement(s) it guards and proceed to whatever follows the if-elif-else.

Equivalent Scripts

```python
x = input('Score: ')  
if x>=90:  
    print 'A'  
elif x>=80:  
    print 'B'  
elif x>=70:  
    print 'C'  
else:  
    print 'U'
```

The one on the left is better. The letter grade is an essential feature of the computation and having a variable that houses it is a reminder of that fact.
Legal Not to Have the "Else"

```python
grade = 'B'
nApples = input('#Apples sent to Prof:')
if nApples<10:
    grade = grade + '-'
print grade
```

Let's review all the "if" variations...

### Standard if-else

```python
if A boolean expression:
    Code that is executed after the whole "if" is processed.
else:
```

Exactly one of the green boxes is executed.

### if-elif

```python
if A boolean expression:
elif Another boolean expression:
```

If both boolean expressions are false, no green box is executed. Otherwise, the "first" green box that is "guarded" by a true boolean expression is executed.

### Multiple if-elif With Else

```python
if:
elif:
elif:
elif:
else:

The first green box guarded by a true boolean expression is executed. If they are all false, then the else green box is executed.

### Multiple if-elif With No Else

```python
if:
elif:
elif:
elif:

Note that if all the boolean expressions are False, no green code is executed. Otherwise, the first green box guarded by a true boolean expression is executed.

### Boolean Operations

<table>
<thead>
<tr>
<th>Rainy</th>
<th>Sunny</th>
<th>Rainbow</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

It is possible to combine two boolean values to get a new boolean value.
### Boolean Operations

This table represents the boolean operations:

<table>
<thead>
<tr>
<th>Sleepy</th>
<th>Tired</th>
<th>Crabby</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

It is possible to combine two boolean values to get a new boolean value.

### The and Operation

Let's consider the following:

- \( x \rightarrow 3 \)
- \( y \rightarrow 6 \)
- \( z \rightarrow 9 \)

The table for the and operation is as follows:

<table>
<thead>
<tr>
<th>( x &lt; y )</th>
<th>( x &lt; z )</th>
<th>( x &gt; y )</th>
<th>( x &gt; z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x &lt; y )</td>
<td>( x &lt; z )</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>( x &gt; y )</td>
<td>( x &lt; z )</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>( x &lt; y )</td>
<td>( x &gt; z )</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

### Example 1

Fact: A length-4 string is a palindrome if the first and last characters are the same and the middle two characters are the same.

Example:

```python
s = input('length-4 string: ')
if (s[0] == s[3]) and (s[1] == s[2]):
    print 'palindrome'
else:
    print 'not a palindrome'
```

### Example 2

Fact: \( x \) is inside the interval \([L,R]\) if it is no smaller than \( L \) and no bigger than \( R \).

```python
x = input('x: ')
L = input('L: ')
R = input('R: ')
if (L <= x) and (x <= R):
    print 'Inside'
else:
    print 'Outside'
```

### Equivalent Solutions

```python
x = input('x: ')
L = input('L: ')
R = input('L: ')
if L <= x <= R:
    print 'Inside'
else:
    print 'Outside'
```
The or Operation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>or</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Here and are Boolean-valued expressions

Example 1

Fact: is inside the interval [L,R] if it is no smaller than L and no bigger than R.

```python
x = input('x: ')  
L = input('L: ')  
R = input('R: ')  
if (x < L) or (R < x):  
    print 'Outside'  
else:  
    print 'Inside'
```

The not Operation

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>not</td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

Here is a boolean-valued expression

Equivalent Solutions

Fact: is inside the interval [L,R] if it is no smaller than L and no bigger than R.

```python
if (x < L) or (R < x):  
    print 'Outside'  
else:  
    print 'Inside'
```

The not Operator

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>not</td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

Here is a boolean-valued expression

A Note on Boolean Variables

Boolean expressions either have the value True or the value False.

When a Boolean expression is evaluated, the result can be stored in a variable, e.g.,

```
outsideInterval = x < L or R < x
```

We say that outsideInterval is a Boolean variable.
**Boolean Variables For Clarity**

```
Y = input('Enter a 4-digit integer: ')
CenturyYear = (Y%100 == 0)
if CenturyYear:
    LeapYear = (Y%400 == 0)
else:
    LeapYear = (Y%4 == 0)
```

Thus, 1960, 2000 and 2400 are leap years. 1961 and 1900 are not. This code assigns the value of True to LeapYear if Y encodes a leap year. It assigns the value of False to LeapYear if Y does not encode a leap year.

---

**A Summarizing Example**

Input a string. If it has even length, then hyphenate in the middle:

- baseball → base-ball

If it has odd length, then hyphenate around the middle character:

- frisbee → fri-s-bee

---

**The len Function**

If ever you need to compute the length of a string then use the built-in function `len`.

```
s = 'abcdef'
len(s)  →  6
```

---

**So Let's Solve this Problem**

Input a string. If it has even length, then hyphenate in the middle:

- baseball → base-ball

If it has odd length, then hyphenate around the middle character:

- frisbee → fri-s-bee

---

**Developing a Solution**

Instead of just showing the solution, let's "derive" the solution using a methodology that is called stepwise refinement.

---

The course is really about problem solving with the computer. So developing problem-solving strategies is VERY IMPORTANT.
"Reformat" the task.
Read in the string
Compute its length
if the length is even
    Hyphenate in the middle
else
    Hyphenate around around the middle character.

Still in English, but it looks a little more like python.

Refine

s = input('Enter a string: ')
n = len(s)
if the length is even
    Hyphenate in the middle
else
    Hyphenate around around the middle character.

We have turned the first two lines into python.

Refine Some More

h = input('Enter a string: ')
n = len(s)
if n%2==0:
    # s has even length
    Hyphenate in the middle
else:
    # s has odd length
    Hyphenate around around the middle character.

We add comments to summarize what we may assume about the value of n.

Refine Some More

Read in the string
Compute its length
if the length is even
    Hyphenate in the middle
else
    Hyphenate around around the middle character.

Turn these into Python.

Refine Some More

h = input('Enter a string: ')
n = len(s)
if n%2==0:
    # s has even length
    Hyphenate in the middle
else:
    # s has odd length
    Hyphenate around around the middle character.

How do we check if the value in n is even?

Refine Some More

h = input('Enter a string: ')
n = len(s)
if n%2==0:
    # s has even length
    Hyphenate in the middle
else:
    # s has odd length
    Hyphenate around around the middle character.

Figure out the even-length hyphenation
Even-Length Hyphenation

We look at a small example.
These statements

\[
\begin{align*}
    s &= 'abcdef' \\
    h &= s[0:3] + '-' + s[3:] \\
    \text{assign} 'abc-def' \text{ to } h.
\end{align*}
\]

In general:

\[
\begin{align*}
    m &= n/2 \\
    h &= s[0:m] + '-' + s[m:]
\end{align*}
\]

Refine Some More

\[
\begin{align*}
    h &= \text{input('Enter a string: ')} \\
    n &= \text{len}(s) \\
    \text{if } n\%2==0: \\
        \# \text{ s has even length} \\
        m &= n/2 \\
        h &= s[0:m] + '-' + s[m:] \\
    \text{else}:
        \# \text{ s has odd length} \\
        \text{Hyphenate around around the middle character.}
\end{align*}
\]

Odd-Length Hyphenation

We look at a small example.
This

\[
\begin{align*}
    s &= 'abcdefg' \\
    h &= s[0:3] + '-' + s[3] + '-' + s[3:] \\
    \text{assigns} 'abc-d-efg' \text{ to } h.
\end{align*}
\]

In general:

\[
\begin{align*}
    m &= n/2 \\
    h &= s[0:m] + '-' + s[m] + '-' + s[m+1:]
\end{align*}
\]

Refine Some More

\[
\begin{align*}
    h &= \text{input('Enter a string: ')} \\
    n &= \text{len}(s) \\
    \text{if } n\%2==0: \\
        \# \text{ s has even length} \\
        m &= n/2 \\
        h &= s[0:m] + '-' + s[m:] \\
    \text{else}:
        \# \text{ s has odd length} \\
        \text{Hyphenate around around the middle character.}
\end{align*}
\]

Done!

\[
\begin{align*}
    h &= \text{input('Enter a string: ')} \\
    n &= \text{len}(s) \\
    \text{if } n\%2==0: \\
        \# \text{ s has even length} \\
        m &= n/2 \\
        h &= s[0:m] + '-' + s[m:] \\
    \text{else}:
        \# \text{ s has odd length} \\
        m &= n/2 \\
        h &= s[0:m] + '-' + s[m] + '-' + s[m+1:]
\end{align*}
\]

Summary

1. A Boolean expression evaluates to either True or False
2. A Boolean expression is made up of comparisons that are either True or False
3. The and, or, not operations combine Boolean values.
4. Various if constructions can be used to organize conditional execution.