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)
```
Motivation

Problem:

Assign positive float values to variables $a$ and $b$ and print the values $a^{**}b$ and $b^{**}a$.

Solution:

$7^{**}2 < 2^{**}7$  $2^{**}3 < 3^{**}2$
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
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.

\[
\begin{align*}
aTob & \rightarrow 128 \\
bToa & \rightarrow 49
\end{align*}
\]
aTob = a**b
bToa = b**a
if aTob > bToa:
    print aTob
else:
    print bToa

The comparison
aTob > bToa
is called a boolean expression.
It is either True or False

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

\[
\begin{align*}
\text{aTob} &= a^{**}b \\
\text{bToa} &= b^{**}a \\
\text{if } \text{aTob} > \text{bToa}: \\
&\quad \text{print } \text{aTob} \\
\text{else}: \\
&\quad \text{print } \text{bToa}
\end{align*}
\]

\[
\begin{align*}
\text{aTob} &\rightarrow 128 \\
\text{bToa} &\rightarrow 49
\end{align*}
\]

The boolean expression \( \text{aTob} > \text{bToa} \) is True so execute \( \text{print } \text{aTob} \).

Is the value of \( \text{aTob} \) larger than the value of \( \text{bToa} \)? Yes!
If-Else: How Does it Work?

\[
a_{\text{Tob}} = a^{**}b \\
b_{\text{Toa}} = b^{**}a
\]

if \(a_{\text{Tob}} > b_{\text{Toa}}\):
    print \(a_{\text{Tob}}\)
else:
    print \(b_{\text{Toa}}\)

Now let's suppose the value of \(a\) is 7 and the value of \(b\) is 2.
If-Else: How Does it Work?

\[
a_{Tob} = a^{**b} \\
b_{Toa} = b^{**a} \\
\text{if } a_{Tob} > b_{Toa}: \\
\quad \text{print } a_{Tob} \\
\text{else:} \\
\quad \text{print } b_{Toa}
\]

Is the value of \( a_{Tob} \) larger than the value of \( b_{Toa} \)?
If-Else: How Does it Work?

\[
\begin{align*}
\text{aTob} &= a**b \\
\text{bToa} &= b**a \\
\text{if } \text{aTob} > \text{bToa}: \\
& \quad \text{print aTob} \\
\text{else:} \\
& \quad \text{print bToa}
\end{align*}
\]

The boolean expression \( \text{aTob} > \text{bToa} \) is False so execute \( \text{print bToa} \).

Is the value of \( \text{aTob} \) larger than the value of \( \text{bToa} \)? No!
If-Else: How Does it Work?

\[
\begin{align*}
a_{Tob} &= a^{**}b \\
b_{Toa} &= b^{**}a \\
\text{if } a_{Tob} > b_{Toa}: \\
&\quad \text{print } a_{Tob} \\
\text{else:} \\
&\quad \text{print } b_{Toa}
\end{align*}
\]

Note the punctuation and the indentation.

This is essential syntax.

Forgetting the colons is a major boo boo!
“Synonym”

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

if a**b > b**a:
    print a**b
else:
    print b**a
```

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

if **Boolean expression**:

Statements to execute if the expression if True

else:

Statements to execute if the expression if False

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

```python
if a**b > b**a:
    z = b**a
else:
    z = a**b

print 'The smaller value is:',z
```

The blue box decides whether the green box or the pink box is executed.

After that choice is processed, this print statement is carried out.
Reminder that Indentation Is Important

if x%2==0:  
y = x/2 
print y
else:
    y = (x+1)/2
print y

if x%2==0:
    y = x/2
print y
else:
    y = (x+1)/2
print y

If x is even, then the code on the left will print the value of x/2 while the code on the right will print the value of x/2 twice (on separate lines).
Another Example

Problem:

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

Solution:

\[ s = s[0:4] + \text{'ies'} \]
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.
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.
A new comparison is being used.

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

Discussion of Solution

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

The print statement is executed after the if-else is processed. E.g.

```
carry   carries
```
Relational Operators

<   Less than
>
Greater than
<=  Less than or equal to
>=  Greater than or equal to
==  Equal to
!=  Not equal to
Relational Operators in Action

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

Comparing for equality...

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

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

Comparing for alphabetical order...

```python
>>> s = 'Dog'
>>> s > 'Horse'
False

>>> s < 'Horse'
True

>>> s < 'dog'
True
```

Alphabetical order. If $s < t$ is true then $s$ comes before $t$ in the “extended dictionary” based on this ordering of characters:

'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
Relational Operators in Action

\[
x \rightarrow \text{'key'} \quad y \rightarrow \text{'hockey'}
\]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x &lt; y )</td>
<td>False</td>
</tr>
<tr>
<td>( x &gt; y )</td>
<td>True</td>
</tr>
<tr>
<td>( \text{'hoc'} + x \leq y )</td>
<td>True</td>
</tr>
<tr>
<td>( x \geq y )</td>
<td>True</td>
</tr>
<tr>
<td>( x == y[3:] )</td>
<td>True</td>
</tr>
<tr>
<td>( x != x + )</td>
<td>True</td>
</tr>
</tbody>
</table>

Comparisons based on alphabetical order.

\( x < y \) is false because 'key' does not come before 'hockey' in the dictionary.
Another Problem

Assume that $s_1$ and $s_2$ are initialized strings.

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

if s1 < s2:
    print s1
    print s2
else:
    print s2
    print s1

s1 --> 'cat'
s2 --> 'dog'

s1 < s2

Is this True or False?
Solution

if s1 < s2:
    print s1
    print s2
else:
    print s2
    print s1

s1 --- > 'cat'
s2 --- > 'dog'

It’s true!

Output:
cat
dog
Solution

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

`s1 < s2` is **True**.

- `s1` --> ‘dog’
- `s2` --> ‘cat’
- `s1 < s2`
Solution

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

```
s1 ---> 'dog'
s2 ---> 'cat'
```

It's false!

Output:
```
cat
dog
```
Indentation Is Important

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

```
s1 --> 'cat'
s2 --> 'dog'
```

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"
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”

Note the punctuation and the indentation.
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.

x ---> 75
If-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.

x ---+ 75
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. Yes.
3. Execute the statement(s) it guards and proceed to whatever follows the if-elif-else

The indentation scheme “tells” Python what comes after the if-elif-else
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. Yes.  
3. Execute the statement(s) it guards and proceed to whatever follows the If-elif-else
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
```

```
x  -->  65
```

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

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
```

1. Is this true?
2. No.
3. Execute “the else”
4. Proceed to what follows the if-elif-else.

$x$ ---$>$ $65$
x = input('Score: ')  
if x>=90:  
    grade = 'A'
elif x>=80:  
    grade = 'B'
elif x>=70:  
    grade = 'C'
else:  
    grade = 'U'
print grade

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”

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**

if \[ A \text{ boolean expression} \] :

else:

**Code that is executed after the whole “if” is processed.**
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

if : 
   
elif : 
   
elif : 
   
else: 
   

The first green box guarded by a true boolean expression is executed. If they are all false, then the else's green box is executed.
Multiple if-elif With No Else

if
    : green
elif
    : green
elif
    : green
elif
    : green

Note that if all the boolean expressions are False, then 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.
It is possible to combine two boolean values get a new boolean value.
The and Operation

\( x \longrightarrow 3 \quad y \longrightarrow 6 \quad z \longrightarrow 9 \)

\( (x < y) \ \text{and} \ (x < z) \quad \text{True} \)
\( (x > y) \ \text{and} \ (x < z) \quad \text{False} \)
\( (x < y) \ \text{and} \ (x > z) \quad \text{False} \)
\( (x > y) \ \text{and} \ (x > z) \quad \text{False} \)
The and Operation

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

Here and are Boolean-valued expressions
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.

```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

```
x = input('x: ')  
L = input('L: ')  
R = input('R: ')  

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

```
x = input('x: ')  
L = input('L: ')  
R = input('R: ')  

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

Here \( \text{and} \) and \( \text{or} \) are Boolean-valued expressions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>or</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 1

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 (x<L) or (R<x):
    print 'Outside'
else:
    print 'Inside'
```
Equivalent Solutions

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

if \((x<L)\) or \((R<x)\):
    print ‘Outside’
else:
    print ‘Inside’

if \((L<=x)\) and \((x<=R)\):
    print ‘Inside’
else:
    print ‘Outside’

Often you can arrange a conditional execution in several ways.
The **not** Operation

\[ x \rightarrow 3 \quad \text{y} \rightarrow 6 \]

\[
\begin{align*}
\text{not (x < y)} & \quad \text{False} \\
\text{not (x > y)} & \quad \text{True}
\end{align*}
\]
The not Operator

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<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`.

```python
s = 'abcdef'

n = len(s)
m = n/2
First = s[:m]
Second = s[m:]
```

- `x` ----> 'abcdef'
- `n` ----> 6
- `m` ----> 3
- `First` ----> 'abc'
- `Second` ----> 'def'
The `len` Function

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

```
s = 'abcdefg'
n = len(s)
m = n/2
First = s[:m]
Second = s[m:]
```

```
x ----> 'abcdefg'
n ----> 7
m ----> 3
First ----> 'abc'
Second ----> 'defg'
```
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.
"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.
Refine

```python
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

```python
s = input(‘Enter a string: ‘)
n = len(s)
if the length is even
    Hyphenate in the middle
else
    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.

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

```python
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

\[
s = 'abcdef'\\
h = s[0:3] + '-' + s[3:]\\assign 'abc-def' to h.
\]

In general:

\[
m = n/2\\h = s[0:m] + '-' + s[m:]
\]
Refine Some More

```python
h = input('Enter a string: ')
n = len(s)
if n%2==0:
    # s has even length
    m = n/2
    h = s[0:m] + '-' + s[m:]
else:
    # s has odd length
    Hyphenate around around the middle character.
```
Refine Some More

```python
h = input('Enter a string: ')  
n = len(s)  
if n%2==0:  
    # s has even length  
    m = n/2  
    h = s[0:m] + '−' + s[m:]  
else:  
    # s has odd length  
    Hyphenate around around the middle character.
```

Figure out the odd-length hyphenation
Odd-Length Hyphenation

We look at a small example.
This

\[
s = 'abcdefg'
\]

\[
h = s[0:3] + '-' + s[3] + '-' + s[3:]
\]

assigns ‘abc-d-efg’ to \( h \).

In general:

\[
m = n/2
\]

\[
h = s[0:m] + '-' + s[m] + '-' + s[m+1:]
\]
h = input(`Enter a string: `)
n = len(s)
if n%2==0:
    # s has even length
    m = n/2
    h = s[0:m] + `-' + s[m:]
else:
    # s has odd length
    m = n/2
    h = s[0:m]+`-'+s[m]+`-'+s[m+1:]
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.