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
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?

Let’s suppose the value of \(a\) is 2 and the value of \(b\) is 7.

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

Solution Using If-Else

```
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

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

Is the value of aTob larger than the value of bToa? Yes!

If-Else: How Does it Work?

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

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

Is the value of aTob larger than the value of bToa? Yes!

If-Else: How Does it Work?

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

Is the value of aTob larger than the value of bToa? No!

If-Else: How Does it Work?

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

In a comparison, we can have general expressions on either side of the "<".

"Synonym"

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

Reminder that Indentation Is Important

```python
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 x/2 while the code on the right will print (x+1)/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:
```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.

Solution

```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
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

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

Boolean Operations with Strings

Are two strings equal?

>>> s = 'abc'
>>> s == 'abc'
True
>>> s == 'abc '
False

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

Alphabetical order?

>>> 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:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z

Relational Operators in Action

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

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

Are two strings equal?
Relational Operators in Action

\begin{align*}
\text{x} & \quad \text{-->} \quad \text{key} \\
\text{y} & \quad \text{-->} \quad \text{hockey}
\end{align*}

\begin{align*}
x < y & \quad \text{False} \\
x > y & \quad \text{True} \\
\text{hoc} + x & \quad \text{<=} \quad y \quad \text{True} \\
x & \quad \text{>=} \quad y \quad \text{True} \\
x & \quad \text{==} \quad y[3:] \quad \text{True} \\
x & \quad \text{!=} \quad x + \text{'}h\text{'} \quad \text{True}
\end{align*}

Comparisons based on alphabetical order.
\text{key} \text{ is false because } \text{key} \text{ does not come before } \text{hockey} \text{ in the dictionary.}

Another Problem

Assume that \text{s1} and \text{s2} are initialized strings.
Write code that prints them in alphabetical order on separate lines.

Solution

\begin{verbatim}
if s1<s2:
    print s1
    print s2
else:
    print s2
    print s1
\end{verbatim}

\text{s1} < \text{s2} 
Is this True or False?

Solution

\begin{verbatim}
if s1<s2:
    print s1
    print s2
else:
    print s2
    print s1
\end{verbatim}

\text{s1} < \text{s2} 
Is this True or False?
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
```

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
```

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
```

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: ')  # x --> 75
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.

---

If-Elif-Else: How it Works

```python
x = input('Score: ')  # x --> 95
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: ')  # x --> 65
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: ')  # x --> 65
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.

---

Equivalent Scripts

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

I prefer the one on the left. The letter grade is an essential feature of the computation and having a variable that houses it reminds me 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 executed
else:
    # Code executed
```

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

Exactly one of the green boxes is executed

### if-elif

```python
if A boolean expression:
    # Code executed
elif Another boolean expression:
    # Code executed
```

*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 A boolean expression:
    # Code executed
elif Another boolean expression:
    # Code executed
elif Another boolean expression:
    # Code executed
else:
    # Code executed
```

*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 A boolean expression:
    # Code executed
elif Another boolean expression:
    # Code executed
elif Another boolean expression:
    # Code executed
elif Another boolean expression:
    # Code executed
```

*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.*

### More Complicated Boolean Expressions

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

*This showcases the and operator.*
The and Operator

<table>
<thead>
<tr>
<th>True</th>
<th>True</th>
<th>True</th>
</tr>
</thead>
<tbody>
<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>

Here and are Boolean-valued expressions.

More Complicated Boolean Expressions

\[
\begin{align*}
(x \lt y) & \lor (x \lt z) & \text{True} \\
(x \gt y) & \lor (x \lt z) & \text{True} \\
(x \lt y) & \lor (x \gt z) & \text{True} \\
(x \gt y) & \lor (x \gt z) & \text{False}
\end{align*}
\]

This showcases the or operator.

Example

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('s: ')  
if (s[0]==s[3]) and (s[1]==s[2]):  
    print 'palindrome'
else:  
    print 'not a palindrome'
```

Example

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 Solution

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

The or Operator

<table>
<thead>
<tr>
<th>True</th>
<th>True</th>
<th>True</th>
</tr>
</thead>
<tbody>
<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
Fact: A length-4 string is a partial palindrome if the first and last characters are the same or if the middle two characters are the same.

```python
s = input('s: ')
if (s[0] == s[3]) or (s[1] == s[2]):
    print 'partial palindrome'
else:
    print 'not a partial palindrome'
```

Example
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'
```

Example
Fact: x 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'
```

More Complicated Boolean Expressions

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

- not \((x < y)\) False
- not \((x > y)\) True

Often you can arrange a conditional execution in several ways.

The not Operator

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

Here \(\text{not}\) is a boolean-valued expression.

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'
x --> 'abcdef'
n --> 6
m = n/2
m --> 3
First = s[:m]
First --> 'abc'
Second = s[m:]
Second --> 'def'
```

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.

"Reformat" the task.

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

```
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

\[
s = \text{input('Enter a string: ')} \\
n = \text{len}(s) \\
\text{if the length is even} \\
\quad \text{Hyphenate in the middle} \\
\text{else} \\
\quad \text{Hyphenate around around the middle character.}
\]

We have turned the first two lines into python.

Refine Some More

\[
s = \text{input('Enter a string: ')} \\
n = \text{len}(s) \\
\text{if the length is even} \\
\quad \text{Hyphenate in the middle} \\
\text{else} \\
\quad \text{Hyphenate around around the middle character.}
\]

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

Refine Some More

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

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

Refine Some More

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

Figure out the even-length hyphenation

Even-Length Hyphenation

We look at a small example.
These statements
\[
s = \text{`abcdef'} \\
h = s[0:3] + '-' + s[3:] \\
\]
assign `abc-def' to h.

In general:
\[
m = n/2 \\
h = s[0:m] + '-' + s[m:] \\
\]

Refine Some More

\[
h = \text{input('Enter a string: ')} \\
n = \text{len}(s) \\
\text{if } n\%2==0: \\
\quad \# \text{s has even length} \\
\quad m = n/2 \\
\quad h = s[0:m] + '-' + s[m:] \\
\text{else:} \\
\quad \# \text{s has odd length} \\
\quad \text{Hyphenate around around the middle character.}
\]
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
    m = n/2
    h = s[0:m] + ' - ' + s[m] + ' - ' + s[m+1:]
```

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:]
```

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.

Terminology

**boolean**

A primitive type whose values are True and False.