Lecture 05

topics:

• Python Strings as **Sequences**
• String **Immutability**
• How to *index* and *slice* sequences
• Slicing Strings
• Branching

• the **if**, **if-else**, **if-elif-else** statements
Review – from last week’s Programming Assignment: Indexing Strings

- Sequential access of sequence elements: Access in order
- Random access: Direct access to any element
- Indexing: Process used to access a specific element of a sequence
- Member: An element of a sequence
- Python allows for random access to sequences (such as strings) via indexing
Review – from last week's Programming Assignment:
Indexing Strings

- Use brackets and position number to index
- Indexing for positive position numbers starts at 0
- Length of sequence minus one is last position
- Attempt to access beyond last position results in error

```python
>>> word = "index"
>>> word[3]
'e'
```
Review – from last week's Programming Assignment:
Indexing Strings

- Can use negative position numbers
- Start at end of sequence with position number: \(-1\)
- End at first element, with position number: negative sequence length

```python
>>> word = "index"
>>> word[-2]
'e'
```
Review – from last week's Programming Assignment: Indexing Strings

Sequence Indexing
String **Immutability**

```python
>>> word = "index"
>>> word[0] = "j"
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    word[0] = "j"
TypeError: 'str' object does not support item assignment
```

- if a type is Mutable: it is "**change-able**"

- if a type is Immutable: "**un-change-able**" as shown in the above example. If we attempt to modify a string by changing one of its letters, we get an error!

  - Strings are **immutable** sequences; they can't be changed

  - But you can create new strings from existing ones: (e.g. with the `concatenation` operation)
...however, we can assign an entirely new value to a string variable. The previous string value is destroyed, and an entirely new string value is created and assigned to the same variable name.

The variable name is originally pointing to the value "Chris". Then the pointer is moved to point to the value "Jackson".
Building a New String with Concatenation

• Python can't modify an existing string.

• as on the previous slide, Python can "build" (create) a new string. Python can also create a new string with the concatenation operator:

```python
>>> name = "Jackson"
>>> name
'Jackson'
>>> name = name + " Jane"
>>> name
'Jackson Jane'
```
side note --- a useful concept: **Constants**
(variables that shouldn't change)

- for example, we could define... `VOWELS = "aeiou"

- **Constant**: in Python it's a variable where **the value** associated with a name is **not meant to be changed**

- Naming convention: use all uppercase variable names

- Can make programs clearer

- Saves retyping (and possibly errors from typos)

- However, there are no true constants in Python:
  (it's a programmer's convention) when you see an all-uppercase variable name, you're just not supposed to change its value!
Creating New Strings from Existing Ones

• `new_message += letter`

• Concatenation creates brand-new string

• Remember, strings are immutable

• So, `new_message` becomes the newly created string resulting from concatenation
Next Topic:
Slicing Strings

- a slice is a copy of a continuous section of a sequence

- Python allows you to:
  - make slices (copies) of continuous sections of sequence elements
  - slice one element or multiple, continuous part of sequence
  - create a slice that is copy of entire sequence
Slicing

- Slicing end points

- An example of slicing *end point numbers* for the string "pizza".
Slicing (continued)

```python
>>> word = "pizza"
>>> word[0:5]
'pizza'
>>> word[1:3]
'iz'
>>> word[-4:3]
'iz'
```
Slicing (continued)

```python
>>> word = "pizza"
>>> word[:0]
''
>>> word[:4]
'pizz'
>>> word[2:]
'zza'
>>> word[:]
'pizza'
```
Next Topics: **Branching** and the **if**... statements

- next topics:
  - conditional execution of code
  - A.K.A. branching
  - can be accomplished with the
    - `if`
    - `if-else`
    - `if-elif-else` statements
Review: Boolean Conditions

• A **Boolean condition** is anything that is either **True** or **False**. Here are some statements that can be either True or False.

• Can you evaluate their True/False value?
  If yes, what is it?
  If not, what is missing for you to be able to determine the True/False value?

  • a variable has the value “test” in it
  • 10 > 5
  • a variable’s value is not 0
# Review: Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Equals</td>
<td><code>5 == 6</code></td>
<td>False</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not Equal</td>
<td><code>5 != 6</code></td>
<td>True</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater Than</td>
<td><code>5 &gt; 6</code></td>
<td>False</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater Than or Equal</td>
<td><code>5 &gt;= 6</code></td>
<td>False</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less Than</td>
<td><code>5 &lt; 6</code></td>
<td>True</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less Than or Equal</td>
<td><code>5 &lt;= 6</code></td>
<td>True</td>
</tr>
</tbody>
</table>

Warning! It’s easy to confuse `=` and `==`. They mean very different things....
# A simple greeting program

# Get the user's name
name = input("Please enter your name: ")

# Get the user's age
age = int(input("Please enter your age: "))

# Display a welcome message
print ("Thanks for using this program," + str(name) + "
". Lookin' good for a " + str(age) + " year old!")
Branching

• So far, sequential programs – no change in behavior!

• What if we ask for the user’s age, and provide a warning if the user is less than 18 years old?

• If the user is less than 18, output:
  • “Don't forget to ask your parents' permission!”

• We’ll use an if statement.
the **if** statement

• the Python code for a basic `if` statement looks like this:

```python
if Boolean Condition :
    # here we put code for...
    # ....whatever we want to happen if the condition is true

# here we continue with code that...
# ... needs to happen all the time
```

• You **must** indent every line of code that goes inside the `if` statement.

• You can use either:
  a. always the same amount of `<space>`, i.e. " ".
      ← **recommended:** 4 (four) " " **space** characters.
  b. always the same amount of `<tab>` characters.
      ← **not recommended.**
the if statement

# our first branching in a program

age = int(input("Please enter your age: "))

# check if under age
if age < 18:
    print("Don't forget to ask your parents' permission!")
Short Greeting Program with Branching

# A simple greeting program

# Get the user's name
name = input("Please enter your name: ")

# Get the user's age
age = int(input("Please enter your age: "))

# check if under age
if age < 18 :
    print ("Don't forget to ask your parents' permission!")

# Display a welcome message
print ("Thanks for using this program, " + str(name) + \\
    ". Lookin' good for a " + str(age) + " year old!")
Branching

• What about multiple checks/tests/conditions?

• For example:
  • “Ask a user to vote for Candidate A, B, or C, and display a different message for each, plus a message if they try to ‘write in’ someone else.”

• We could do this with multiple if statements, but we’re going to use something new – the **elif** statement (short for else if) and the **else** statement.
the if elif else statement

if Boolean Condition :
  # Whatever we want to happen if this condition is true

elif Other Boolean Condition :
  # Whatever we want to happen if this condition is true

elif Third Boolean Condition :
  # Whatever we want to happen if this condition is true

else :
  # This will happen only if ALL conditions are false!

  # Something that needs to happen all the time

(side note: in other programming languages statements like these are sometimes called “if-then-else” statement)
Voting Program

# A simple voting program

vote = input("Please vote for Candidate A, B, or C (enter A, B, or C): ")

if vote == 'A' : # notice we use == here!
    print ("Candidate A thanks you for your support.")
elif vote == 'B' :
    print ("Candidate B pledges to make things better.")
elif vote == 'C' :
    print ("Candidate C promises to stop things from getting worse.")
else :
    print ("Your write-in vote has been discarded.")

print ("Thanks for voting!")
The general structure stops at the first "True" match.
The if-elif-else Structure

one (and only one) of the four possible choices will be True, and only the corresponding statements will be evaluated.

note: in this sample code, statements are missing in the first three cases: if you try and run this code unmodified, you'll get an error! (a comment doesn't count as a statement)

you'll need to write at least something (such as the print statement inside the fourth block) inside each one of the four possible blocks. If you don't want Python to do anything in one of these cases, you may use the pass statement, which ...doesn't do anything!

```python
if mood == 1:
    # happy
elif mood == 2:
    # neutral
elif mood == 3:
    # sad
else:
    print("Illegal mood value!")
```
The if-elif-else Structure (continued)

- the code on the previous page tests a chain of conditions after if and elif (short for “else if”).
- On the first True condition, the associated block of code gets executed, and the entire structure is exited.
- If no condition evaluates to True, the block of code following the (optional) else clause gets executed.
## Branching Structures

### Table 3.2 Branching Structures Summary

<table>
<thead>
<tr>
<th>Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if &lt;condition&gt;: &lt;block&gt;</code></td>
<td><em>if structure. If <code>&lt;condition&gt;</code> is true, <code>&lt;block&gt;</code> is executed; otherwise it's skipped.</em></td>
</tr>
<tr>
<td><code>if &lt;condition&gt;: &lt;block 1&gt;</code></td>
<td><em>if-else structure. If <code>&lt;condition&gt;</code> is true, <code>&lt;block 1&gt;</code> is executed; otherwise <code>&lt;block 2&gt;</code> is executed.</em></td>
</tr>
<tr>
<td><code>else: &lt;block 2&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>if &lt;condition 1&gt;: &lt;block 1&gt;</code></td>
<td><em>if-elif-else structure. The block of the first true condition is executed. If no condition is true, the optional else clause’s block, <code>&lt;block N+1&gt;</code>, is executed.</em></td>
</tr>
<tr>
<td><code>elif &lt;condition 2&gt;: &lt;block 2&gt;</code></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td><code>elif &lt;condition N&gt;: &lt;block N&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>else: &lt;block N+1&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

### If-Elif-Else: General Structure

[Diagram of if-elif-else statement flowchart]