Python syntax and semantics

• Definitions
• Example programs
• If we have time, we’ll design one instruction of an ISA 😊
• Homeworks can be collected at end of class
Assignments

- \( x = 2 \)
- \( x = 7 * x \)
- \( x = 3 * x + x \)
## Illegal variable names

<table>
<thead>
<tr>
<th>and</th>
<th>if</th>
<th>from</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>import</td>
<td>yield</td>
</tr>
<tr>
<td>assert</td>
<td>in</td>
<td>global</td>
</tr>
<tr>
<td>break</td>
<td>is</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>lambda</td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>not</td>
<td></td>
</tr>
<tr>
<td>def</td>
<td>or</td>
<td></td>
</tr>
<tr>
<td>del</td>
<td>pass</td>
<td></td>
</tr>
<tr>
<td>elif</td>
<td>print</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td>raise</td>
<td></td>
</tr>
<tr>
<td>except</td>
<td>return</td>
<td></td>
</tr>
<tr>
<td>exec</td>
<td>try</td>
<td></td>
</tr>
<tr>
<td>finally</td>
<td>while</td>
<td></td>
</tr>
<tr>
<td>for</td>
<td>with</td>
<td></td>
</tr>
</tbody>
</table>
Expression

• Valid expressions
  - \( x = 4 \)
  - \( y = x + 1 \)
  - \( x = 2 \)
  - \( x = 5x + 2(3 - y) \)

• NOT VALID
  - "Hello" + 2
  - "hello" * "aerodrome"
String concatenation

- Valid addition of strings
  - “hello” + “world”
Strings

- Things within “ “
- Use \” to represent quotation marks itself
- \n represents a new line (also called line-break)
If/else

```python
if(test):
    indented lines
else:
    indented lines
```

- Else is optional
tests

- expression comparison expression
  - `x == "hello"`
  - `2*x > 5`
  - `x+y != z`

<table>
<thead>
<tr>
<th>Comparison operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
What does this do?

if(1 == 1):
    print("hello")
else:
    print("something is wrong with the universe")
while

while(test):
    lines that you want to run for as long as the test holds true

(the test is re-run every time we get to the end of the indented lines)
counter = 0
while(counter < 3):
    print("(:)")
    counter = counter + 1
print("(:/:)"")
<table>
<thead>
<tr>
<th>counter = 0</th>
<th>It was the next line</th>
<th>Variables after current line runs</th>
<th>Output so far</th>
</tr>
</thead>
<tbody>
<tr>
<td>while(counter &lt; 3):</td>
<td>It was the next line</td>
<td>counter = 0</td>
<td></td>
</tr>
<tr>
<td>print(&quot;.&quot;):&quot;</td>
<td>counter was &lt; 3</td>
<td>counter = 0</td>
<td>:)</td>
</tr>
<tr>
<td>counter = counter + 1</td>
<td>It was the next line</td>
<td>counter = 1</td>
<td>:)</td>
</tr>
<tr>
<td>print(&quot;.&quot;):&quot;</td>
<td>We reached the end of the while block, but counter was &lt; 3, so we go back to the start</td>
<td>counter = 1</td>
<td>:) :)</td>
</tr>
<tr>
<td>counter = counter + 1</td>
<td>It was the next line</td>
<td>counter = 2</td>
<td>:) :)</td>
</tr>
<tr>
<td>print(&quot;.&quot;):&quot;</td>
<td>We reached the end of the while block, but counter was &lt; 3, so we go back to the start</td>
<td>counter = 2</td>
<td>:) :) :)</td>
</tr>
<tr>
<td>counter = counter + 1</td>
<td>It was the next line</td>
<td>counter = 3</td>
<td>:) :) :)</td>
</tr>
<tr>
<td>print(&quot;.&quot;:/&quot;</td>
<td>counter was not &lt; 3</td>
<td>counter = 3</td>
<td>:) :) :) :)</td>
</tr>
</tbody>
</table>
Comments

# this is a comment

Begin line with #
Factorial (ver 1)

print(1*2*3*4*5*6*7*8*9*10)
Factorial (ver 2)

• Step 1: Store the number whose factorial we want in a variable called input.
• Step 2: Store the number 1 in the variable answer.
• Step 3: Store the number 2 in the variable counter.
• Step 4: Repeat the following until counter is greater than input:
  – Substep 1: Replace answer with answer * counter
  – Substep 2: Increment counter by 1
• Step 5: print the number stored in the variable answer.
Factorial (Ver 2)

input = 10
answer = 1
counter = 2
while(counter < input):
    answer = answer * counter
    counter = counter + 1
print(answer)
Summary

- Programs execute one line at a time

```
Assignment
variable name = expression
```
```
Print
print(expression)
```
```
Flow-Control
if or while
```
Next Class

• Read 2.5 through 2.6
• Look at the Online HW system
Announcements
Homework 1 can be collected at end of class.
Only half of the problems of the homework will be graded.
Should have finished reading chapter 2.4.

Python Syntax and Definitions
Assignments
variable_name = expression
Examples:

    x = 2
    x = 7 * x
    x = 3 * x + x

variable_names
    can only contain letters, numbers, and underscores
    cannot begin with a number nor be a reserved word (e.g. if, while, else)
Illegal variable names
as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, and etc.

Expressions
variable_name combined with arithmetic operation
Examples

    x = 4
    y = x + 1
    x = 2
    x = 5 * x + 2 * (3 - y)

Not valid
"hello" + 2
"hello" * "aerodrome"

String concatenation is valid
"hello" + "world"

Strings
Things within ""
Use \" to represent quotation marks itself
Use \n to represent a new line AKA line break
Use \\ to represent a backslash

If/else

if(test):
    indented lines
else:
    indented lines

Else is optional

Tests
Expression comparison expression
Example tests

    x == "hello"
    2 * x > 5
    x + y != z

Example operators
    <, >, <=, >=, ==, !=
while (test):
    lines that you want to run while the test holds true
The rest is re-run every time we get to the end of the indented lines

Comments
    # this is a comment
    Comments begin with #

Example Programs
Example 1
    if(1==1):
        print("hello")
    else:
        print("something is wrong with the universe")
This will print "hello"

Example 2
    counter = 0
    while(counter < 3):
        print(":\)
        counter = counter + 1
    print(":/")

<table>
<thead>
<tr>
<th>Current line</th>
<th>Why we went to this line</th>
<th>Variables after line runs</th>
<th>Output so far</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter = 0</td>
<td>it was the first/next line</td>
<td>counter = 0</td>
<td></td>
</tr>
<tr>
<td>while(counter &lt; 3):</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(&quot;:&quot;)</td>
<td>counter &lt; 3</td>
<td>counter = 1</td>
<td>:)</td>
</tr>
<tr>
<td>counter = counter + 1</td>
<td></td>
<td></td>
<td>:)</td>
</tr>
<tr>
<td>print(&quot;:&quot;)</td>
<td></td>
<td></td>
<td>:)</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(&quot;:/&quot;)</td>
<td>broke out of while loop because counter not less than 3</td>
<td>counter = 3</td>
<td>:)</td>
</tr>
</tbody>
</table>

Example 3
    print(1*2*3*4*5*6*7*8*9*10)
This can be broken into multiple steps by looping and using a counter:
    input = 10
    answer = 1
    counter = 2
    while(counter < input):
        answer = answer * counter
        counter = counter + 1
    print(answer)
Summary

Programs execute one line at a time
Every line is an assignment, a print, or a flow-control

Design 1 Instruction of an ISA (time permitting)

Play the role of machine designer, not compiler
Instruction has an operation, 2 inputs, and 1 output
Need legal names for inputs and outputs
Let’s say the only inputs allowed are r0 to r15

Need a name to set input = 10
  
equals r1, 10 # name can be anything, so use equal
equals r1, 10 # use , instead of <- for compatibility
eq r1, 10 # eq is operation, r1 is destination, 10 is source
ld r1, 10 # replace eq with ld (load) to avoid comparison confusion

Another example for multiplication:
  
mul r2, r2, r3