Python

Types and Conditional Logic
x = 10
y = x
x = 5

What is the value of y?

A 10
B 5
Question

\begin{align*}
x &= 10 \\
y &= x \\
x &= 5
\end{align*}

What is the value of \(y\)?

A 10 ⭐
B 5
Programming
Components

- **Literals**—4
- **Operators**—+
- **Variables**—x
- **Keywords**—import
- **Expressions**—4 + x
- **Statements**—y = 4 + x
How Assignment Works

```python
x = 10
y = x * x
x * x = y  # a neat trick to swap two variables
x, y = y, x
```

Programming
Natural numbers

$\mathbb{N}$

1, 2, 3, 4, 5, ...
Numbers

\[ \mathbb{N}_0 \]

0, 1, 2, 3, 4, 5, ...

whole numbers

Data Types
Numbers

\[ \mathbb{Z} \]

..., $-4$, $-3$, $-2$, $-1$, $0$, $+1$, $+2$, $+3$, ...

integers
Numbers

Data Types
rational numbers
Numbers

$\mathbb{R}$

$\pi, e, 10^{100}, + \frac{1}{10}, 0.25, ...$

real numbers
Numbers

\( \mathbb{C} \); \( i \); \( 1 + i \); \[ \ldots \]

complex numbers
Numbers

\[ \mathbb{C} \]

\[ i, 1 + i, \ldots \]

complex numbers
Python supports several basic number types:

- integer
- float
- complex
Numbers in Python

Python supports several basic number types:

- integer $\Rightarrow \mathbb{Z}$
- float $\Rightarrow \mathbb{R}$ or maybe $\mathbb{Q}$
- complex $\Rightarrow \mathbb{C}$ (again, maybe)
Floating-point numbers include a fractional part.

(Anything with a decimal point—2.4, 3.0.)

Complex is two floats together.

0 + 1j  # "i"
1 + 0j  # "1"
How do binary numbers work?

- Numeric types can be represented in binary:
  - 000 0
  - 010 2
  - 100 4
  - 110 6
  - 001 1
  - 011 3
  - 101 5
  - 111 7

Data Types
How do binary numbers work?

- Numeric types can be represented in binary:
  000 0 010 2 100 4 110 6
  001 1 011 3 101 5 111 7

- Basically, in decimal:
  \[513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0\]
How do binary numbers work?

- Numeric types can be represented in binary:
  
<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>

- Basically, in decimal:

\[ 513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0 \]

- Similarly, in binary:

\[ 1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]
How do binary numbers work?

- Numeric types can be represented in binary:
  
  0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 1 1 0 6
  
  0 0 1 1 0 0 1 1 0 1 0 1 1 1 1 7

- Basically, in decimal:

  \[ 513 = 5 \times 10^2 + 1 \times 10^1 + 3 \times 10^0 \]

- Similarly, in binary:

  \[ 1011_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]

  \[ = 8 + 0 + 2 + 1 = 11 \]
How do binary numbers work?

- But there are only so many bytes in a number!

  11100110 00001111 10000111 11100111
How do binary numbers work?

But there are only so many bytes in a number!

11100110 00001111 10000111 11100111

This represents a limit on the representation of certain numbers.
If we add too much, the number may overflow.
Python integers can be arbitrarily large.
How do binary numbers work?

- Python integers can be arbitrarily large.
  
  \[
  10 \times 100 \\
  10 \times (10 \times 5)
  \]
How do binary numbers work?

- Python integers can be arbitrarily large.
  10 ** 100
  10 ** (10 ** 5)
- Floating-point numbers have limits, though.

Data Types
How do binary numbers work?

- Python integers can be arbitrarily large.
  
  \[
  10 \times 100 \\
  10 \times (10 \times 5)
  \]

- Floating-point numbers have limits, though.
  
  \[
  1.0 \times 10 \times 300 \quad \# \text{ okay} \\
  1.0 \times 10 \times 340 \quad \# \text{ ‘infinite’} \\
  1.0 \times 10 \times -400 \quad \# \text{ ‘zero’}
  \]
How do binary numbers work?

- Python integers can be arbitrarily large.
  
  \[
  10 \times 100 \\
  10 \times (10 \times 5)
  \]

- Floating-point numbers have limits, though.
  
  \[
  1.0 \times 10 \times 300 \quad \# \text{okay} \\
  1.0 \times 10 \times 340 \quad \# \text{‘‘infinite’’} \\
  1.0 \times 10 \times -400 \quad \# \text{‘‘zero’’}
  \]
- Floating-point numbers include a fractional part.
  (Anything with a decimal point—2.4, 3.0.)
- What are the limits?
  - Overflow/underflow (values too big or too small)
  - Arbitrary precision ($\pi$, $e$)
What is an encoding?

What does a binary data value like this represent?

- What does binary data represent?
- How does the processor know?
What is an encoding?

01001000 01000101 01001100 01001100
What does a binary data value like this represent?

- What does binary data represent?
- How does the processor know?
- The **encoding** interprets the value.
What is a **data type**?

- A **data type** defines an encoding rule.
- All values have a type.
What is a **data type**?

- A **data type** defines an encoding rule.
- All values have a type.
- The type defines how data is represented in memory.
- The type defines allowed operations and how they work.
Operators
Evaluating an expression of integers will generally result in an integer answer

3 + 5
Evaluating an expression of integers will generally result in an integer answer

- \( 3 + 5 \)
- EXCEPTION: DIVISION!
Integer operations

- Evaluating an expression of integers will generally result in an integer answer
  - $3 + 5$
  - EXCEPTION: DIVISION!
  - $3 / 4 \rightarrow 0.75$
Evaluating an expression of integers will generally result in an integer answer

- $3 + 5$
- **EXCEPTION: DIVISION!**
- $3 \div 4 \rightarrow 0.75$
- $3 \mathbf{\div} 4 \rightarrow 0$ (floor division)
Evaluating an expression of floating-point values will result in a floating-point answer.
Evaluating an expression of floating-point values will result in a floating-point answer.

- $3.0 + 5.5 \rightarrow 8.5$
Evaluating an expression of floating-point values will result in a floating-point answer.

- $3.0 + 5.5 \rightarrow 8.5$
- $3.0 + 5.0 \rightarrow 8.0$
Floating-point operations

- Evaluating an expression of floating-point values will result in a floating-point answer.
  - $3.0 + 5.5 \rightarrow 8.5$
  - $3.0 + 5.0 \rightarrow 8.0$
  - $3 + 5.5 \rightarrow ?$ (what happens here?)

Engineers and scientists need to think carefully about the precision of answers.
Floating-point operations

- Evaluating an expression of floating-point values will result in a floating-point answer.
  - $3.0 + 5.5 \rightarrow 8.5$
  - $3.0 + 5.0 \rightarrow 8.0$
  - $3 + 5.5 \rightarrow ?$ (what happens here?)

- Engineers and scientists need to think carefully about the precision of answers.
If one type is inadequate for a result, Python “promotes” the result.

1 / 3  # int => float
(-1) ** 0.5  # int => complex
(-1.0) ** 0.5 # float => complex

What happens to -1 ** 0.5?
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )

What is printed to the screen?
x = 4
y = 3 + 1j
z = 33.3333
print( x + y + z )

What is printed to the screen?
A 40
B 40.3333
C 40.3333 + 1j
D None of the above
Question

\begin{align*}
x &= 4 \\
y &= 3 + 1j \\
z &= 33.3333 \\
\text{print}( x + y + z )
\end{align*}

What is printed to the screen?

A 40
B 40.3333
C 40.3333 + 1j
D None of the above
Attribute operator.

- Attribute operator ..
- Reaches inside of a value to access part of its data (called an attribute).
- Extracts special variables stored “inside” of the type.
  
  ```python
  print(x.real)
  print(x.imag)
  ```
Attribute operator:

- Reaches inside of a value to access part of its data (called an attribute).
- Extracts special variables stored “inside” of the type.

```python
print(x.real)
print(x.imag)
```

- Both of these components are floats.
x = (3.5 + 1j)
y = 1
z = x + y

What is the type of z.imag?
x = (3.5 + 1j)
y = 1
z = x + y

What is the type of \texttt{z.imag}?

A int
B float
C complex
x = (3.5 + 1j)
y = 1
z = x + y

What is the type of \texttt{z.imag}?

A int
B float \texttt{⋆z is complex, not its components!}
C complex
x = (3.5 + 1j)
y = 1
z = x + y

What is the value of z.imag?
A 4.5 + 1j
B 4.5
C 1j
D 1.0
x = (3.5 + 1j)
y = 1
z = x + y

What is the value of z.imag?
A 4.5 + 1j
B 4.5
C 1j
D 1.0
Python offers many libraries to support other operations.

```python
import math
math.factorial( 5 )
math.log( 10 )
math.pi
math.e
```
Python offers many libraries to support other operations.

```python
import math
math.factorial(5)
math.log(10)
math.pi
math.e
```

Note that you need to include the module name and the attribute operator `.`.

Langtangen refers to these frequently.
Alternatively, you can retrieve one thing (name) from a module:

```python
from math import pi
from math import log
log(10)
from math import factorial
factorial(5)
```
String Data Type
How does text work?

- Each symbol is stored individually, one byte long:
  - 01001000  72
  - 01000101  69
  - 01001100  76
  - 01001100  76
  - 01001111  79
### ASCII encoding table

<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
<th>ASCII Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(nul)</td>
<td>016</td>
<td>(dle)</td>
<td>032</td>
<td>sp</td>
<td>048</td>
<td>(b)</td>
<td>064</td>
<td>(a)</td>
</tr>
<tr>
<td>001</td>
<td>(soh)</td>
<td>017</td>
<td>(dc1)</td>
<td>033</td>
<td>!</td>
<td>049</td>
<td>1</td>
<td>065</td>
<td>A</td>
</tr>
<tr>
<td>002</td>
<td>(stx)</td>
<td>018</td>
<td>(dc2)</td>
<td>034</td>
<td>&quot;</td>
<td>050</td>
<td>2</td>
<td>066</td>
<td>B</td>
</tr>
<tr>
<td>003</td>
<td>(etx)</td>
<td>019</td>
<td>(dc3)</td>
<td>035</td>
<td>#</td>
<td>051</td>
<td>3</td>
<td>067</td>
<td>C</td>
</tr>
<tr>
<td>004</td>
<td>(eot)</td>
<td>020</td>
<td>(dc4)</td>
<td>036</td>
<td>$</td>
<td>052</td>
<td>4</td>
<td>068</td>
<td>D</td>
</tr>
<tr>
<td>005</td>
<td>(enq)</td>
<td>021</td>
<td>(nak)</td>
<td>037</td>
<td>%</td>
<td>053</td>
<td>5</td>
<td>069</td>
<td>E</td>
</tr>
<tr>
<td>006</td>
<td>(ack)</td>
<td>022</td>
<td>(syn)</td>
<td>038</td>
<td>&amp;</td>
<td>054</td>
<td>6</td>
<td>070</td>
<td>F</td>
</tr>
<tr>
<td>007</td>
<td>(bel)</td>
<td>023</td>
<td>(etb)</td>
<td>039</td>
<td>'</td>
<td>055</td>
<td>7</td>
<td>071</td>
<td>G</td>
</tr>
<tr>
<td>008</td>
<td>(bs)</td>
<td>024</td>
<td>(can)</td>
<td>040</td>
<td>(</td>
<td>056</td>
<td>8</td>
<td>072</td>
<td>H</td>
</tr>
<tr>
<td>009</td>
<td>(tab)</td>
<td>025</td>
<td>(em)</td>
<td>041</td>
<td>)</td>
<td>057</td>
<td>9</td>
<td>073</td>
<td>I</td>
</tr>
<tr>
<td>010</td>
<td>(lf)</td>
<td>026</td>
<td>(eof)</td>
<td>042</td>
<td>*</td>
<td>058</td>
<td>:</td>
<td>074</td>
<td>J</td>
</tr>
<tr>
<td>011</td>
<td>(vt)</td>
<td>027</td>
<td>(esc)</td>
<td>043</td>
<td>+</td>
<td>059</td>
<td>;</td>
<td>075</td>
<td>K</td>
</tr>
<tr>
<td>012</td>
<td>(np)</td>
<td>028</td>
<td>(fs)</td>
<td>044</td>
<td>,</td>
<td>060</td>
<td>&lt;</td>
<td>076</td>
<td>L</td>
</tr>
<tr>
<td>013</td>
<td>(cr)</td>
<td>029</td>
<td>(gs)</td>
<td>045</td>
<td>-</td>
<td>061</td>
<td>=</td>
<td>077</td>
<td>M</td>
</tr>
<tr>
<td>014</td>
<td>(so)</td>
<td>030</td>
<td>(rs)</td>
<td>046</td>
<td>.</td>
<td>062</td>
<td>&gt;</td>
<td>078</td>
<td>N</td>
</tr>
<tr>
<td>015</td>
<td>(si)</td>
<td>031</td>
<td>(us)</td>
<td>047</td>
<td>/</td>
<td>063</td>
<td>?</td>
<td>079</td>
<td>O</td>
</tr>
</tbody>
</table>

**String Data Type**

- **ASCII encoding**
- **Character set**
- **Decimal values**
- **Hexadecimal values**
- **Printable characters**
- **Control characters**
### ASCII encoding table

<table>
<thead>
<tr>
<th>Octet</th>
<th>Character</th>
<th>Value</th>
<th>ASCII Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(null)</td>
<td>016</td>
<td>▶ (dte)</td>
</tr>
<tr>
<td>001</td>
<td>(soh)</td>
<td>017</td>
<td>▷ (dc1)</td>
</tr>
<tr>
<td>002</td>
<td>(stx)</td>
<td>018</td>
<td>‹ (dc2)</td>
</tr>
<tr>
<td>003</td>
<td>(etx)</td>
<td>019</td>
<td> (dc3)</td>
</tr>
<tr>
<td>004</td>
<td>(eot)</td>
<td>020</td>
<td> (dc4)</td>
</tr>
<tr>
<td>005</td>
<td>(enq)</td>
<td>021</td>
<td>$ (nak)</td>
</tr>
<tr>
<td>006</td>
<td>(ack)</td>
<td>022</td>
<td>– (syn)</td>
</tr>
<tr>
<td>007</td>
<td>(bel)</td>
<td>023</td>
<td>‹ (etb)</td>
</tr>
<tr>
<td>008</td>
<td>(bs)</td>
<td>024</td>
<td>↑ (can)</td>
</tr>
<tr>
<td>009</td>
<td>(tab)</td>
<td>025</td>
<td>↓ (em)</td>
</tr>
<tr>
<td>010</td>
<td>(lf)</td>
<td>026</td>
<td>* (eof)</td>
</tr>
<tr>
<td>011</td>
<td>(vt)</td>
<td>027</td>
<td>← (esc)</td>
</tr>
<tr>
<td>012</td>
<td>(np)</td>
<td>028</td>
<td>↓ (fs)</td>
</tr>
<tr>
<td>013</td>
<td>(cr)</td>
<td>029</td>
<td>⇔ (gs)</td>
</tr>
<tr>
<td>014</td>
<td>(so)</td>
<td>030</td>
<td>▲ (rs)</td>
</tr>
<tr>
<td>015</td>
<td>(si)</td>
<td>031</td>
<td>▼ (us)</td>
</tr>
</tbody>
</table>

**Example string**

72 69 76 76 79 = HELLO
### ASCII encoding table

<table>
<thead>
<tr>
<th></th>
<th>String Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(nul) 016 ▲ (dle) 032 sp 048 ə 064 ® 080 P 096 ` 112 p</td>
</tr>
<tr>
<td>001</td>
<td>(soh) 017 ▲ (dc1) 033 ! 049 1 065 A 081 Q 097 a 113 q</td>
</tr>
<tr>
<td>002</td>
<td>(stx) 018 ₡ (dc2) 034 &quot; 050 2 066 B 082 R 098 b 114 r</td>
</tr>
<tr>
<td>003</td>
<td>(etx) 019 !! (dc3) 035 # 051 3 067 C 083 S 099 c 115 s</td>
</tr>
<tr>
<td>004</td>
<td>(eot) 020 ☞ (dc4) 036 $ 052 4 068 D 084 T 100 d 116 t</td>
</tr>
<tr>
<td>005</td>
<td>(enq) 021 $ (nak) 037 % 053 5 069 E 085 U 101 e 117 u</td>
</tr>
<tr>
<td>006</td>
<td>(ack) 022 ‼ (syn) 038 &amp; 054 6 070 F 086 V 102 f 118 v</td>
</tr>
<tr>
<td>007</td>
<td>(bel) 023 ‼ (etb) 039 ' 055 7 071 G 087 W 103 g 119 w</td>
</tr>
<tr>
<td>008</td>
<td>(bs) 024 ‭ (can) 040 ( 056 8 072 H 088 X 104 h 120 x</td>
</tr>
<tr>
<td>009</td>
<td>(tab) 025 ↓ (em) 041 ) 057 9 073 I 089 Y 105 i 121 y</td>
</tr>
<tr>
<td>010</td>
<td>(lf) 026 EOF (eof) 042 * 058 : 074 J 090 Z 106 j 122 z</td>
</tr>
<tr>
<td>011</td>
<td>(vt) 027 ← (esc) 043 + 059 ; 075 K 091 ] 107 k 123 {</td>
</tr>
<tr>
<td>012</td>
<td>(np) 028 L (fs) 044 , 060 &lt; 076 L 092 \ 108 l 124</td>
</tr>
<tr>
<td>013</td>
<td>(cr) 029 ➔ (gs) 045 - 061 = 077 M 093 } 109 m 125 }</td>
</tr>
<tr>
<td>014</td>
<td>(so) 030 ▲ (rs) 046 . 062 &gt; 078 N 094 ^ 110 n 126 ~</td>
</tr>
<tr>
<td>015</td>
<td>(si) 031 ▼ (us) 047 / 063 ? 079 O 095 _ 111 o 127 µ</td>
</tr>
</tbody>
</table>

72 69 76 76 79 = H E L L O

’HELLO’
- As a literal: text surrounded by quotes (single or double).
  - "DEEP"
Strings

- As a literal: text surrounded by quotes (single or double).
  - "DEEP"
- Each symbol is a character.
Strings

- As a literal: text surrounded by quotes (single or double).
  - "DEEP"
- Each symbol is a character.
- Unlike numeric types, strings vary in length.
String operations

- **Concatenation**: combine two strings
  - Uses the `+` symbol
  - `'RACE' + 'CAR'`
String operations

- **Concatenation**: combine two strings
  - Uses the + symbol
  - ’RACE’ + ’CAR’

- **Repetition**: repeat a string
  - Uses the *
  - ’HELLO’ *10
Indexing operator

- Extracts single character or a range of characters
  
a = "FIRE"
  
a[0]
Indexing operator

- Extracts single character or a range of characters
  
a = "FIRE"
a[0]
- The integer is the *index*.
- We count from zero!
- If *negative*, counts down from end.
Slicing operator:

- Extracts range of characters (*substring*)
- Range specified inside of indexing operator

```python
a = "FIREHOUSE"
a[0:4]
```
Slicing operator:

- Extracts range of characters (*substring*)
- Range specified inside of indexing operator
  
  ```
  a = "FIREHOUSE"
  a[0:4]
  ```
- Can be a bit tricky at first:
  - Includes character at first index
  - Excludes character at last index
alpha = "ABCDE"
x = alpha[1:3]

What is the value of x?
A  'AB'
B  'ABC'
C  'BC'
D  'BCD'
E  'CD'
alpha = "ABCDE"
x = alpha[1:3]

What is the value of x?

A  'AB'
B  'ABC'
C  'BC' ⋆
D  'BCD'
E  'CD'
Boolean Logic
bool is a type with two possible values:
  True
  False
We use these to make decisions.
The logic is based on *Boolean algebra*. 
**bool** is a type with two possible values:
- True
- False

We use these to make decisions.

The logic is based on *Boolean algebra*.

**Operators:**
- and
- or
- not
Example: Boolean logic

\[0 < x \leq 10\]

\[(x > 0) \text{ and } (x \leq 10)\]
Example: Boolean logic

\[ 0 < x \leq 10 \]

\((x > 0) \text{ and } (x \leq 10)\)

\(\text{NOT } x > 0 \text{ and } <= 10\)!
Boolean operators

Operators:
- **and**—True only if both sides are True
- **or**—True if either side is True
- **not**—swaps False and True
x = (True and False) and not (True or False)
What is the value of x?
A True
B False
x = (True and False) and not (True or False)
What is the value of x?
x = (False) and not (True)
x = (True and False) and not (True or False)
What is the value of x?
x = (False) and not (True)
x = (False) and (False)
$x = (\text{True and False}) \text{ and not } (\text{True or False})$

What is the value of $x$?

A True

B False
Comparison operators

These produce Boolean output.
- less than, <
- greater than, >
- less than or equal to, <=
- greater than or equal to, >=
- equal to, ==
- not equal to, !=
a = 5
b = 3

x = (a < 5) and ((b <= 5) or (a != b))

What is the value of x?
A True
B False
Example

\[
\begin{align*}
a &= 'URSA MAJOR' \\
b &= 'GEMINI' \\
x &= a < b \text{ and } a[1] \neq b[-2]
\end{align*}
\]

What is the value of \(x\)?

A True
B False

\[\text{True}\]
def fun(a,b):
    return a<b

a = 3
b = 4
x = fun(b,a)

What is the value of x?
A True
B False ∗
Conditional Execution
Control flow represents actual sequence of lines executed by processor.

Conditional execution lets you execute (or not) a block of code based on logical comparison.
Example: if statement

```python
ans = -5
if ans < 0:
    print( "The number was negative." )
```
We create an if statement as follows:
- the keyword if
- a logical comparison (results in bool)
- a indented **block** of code
Alternative execution

- This lets us make decisions in the program!
- We can change program behavior as it executes.
Example: if statements

```
a
ans = ( -15 / 3 ) + 10
if ans < 0:
    print( "The number was negative." )
if ans > 0:
    print( "The number was positive." )
if ans == 0:
    print( "The number was zero." )```
h = '2'
j = '1'
k = int( j+h ) % 5
if ( k % 3 ) == 2:
    k = k ** 3
if ( k % 3 ) == 1:
    k = k ** 2
if hour < 12:
    print( "morning" )
else:
    if hour < 18:
        print( "afternoon" )
    else:
        print( "nighttime" )
if hour < 12:
    print( "morning" )
elif hour < 18:
    print( "afternoon" )
else:
    print( "nighttime" )
Conditional Execution

- We create an if/elif/else statement as follows:
  - the keyword if
  - a logical comparison (results in bool)
  - a block of code
  - the keyword elif
  - a logical comparison (results in bool)
  - a block of code
  - the keyword else
  - a different block of code
Example

```python
if day == 1:
    print("Sunday")
else:
    if day == 2:
        print("Monday")
    else:
        if day == 3:
            print("Tuesday")
        else:
            if day == 4:
                print("Wednesday")
            else:
                if day == 5:
                    print("Thursday")
                else:
                    if day == 6:
                        print("Friday")
                    else:
                        if day == 7:
                            print("Saturday")
                        else:
                            print("Unknown day")
```

Conditional Execution
Example

```python
if day == 1:
    print("Sunday")
elif day == 2:
    print("Monday")
elif day == 3:
    print("Tuesday")
elif day == 4:
    print("Wednesday")
elif day == 5:
    print("Thursday")
elif day == 6:
    print("Friday")
elif day == 7:
    print("Saturday")
else:
    print("That is not a valid day.")
```
Compare these two:

```python
x = 5

if x < 6:
    print(x)
if x < 7:
    print(x)

if x < 6:
    print(x)
elif x < 7:
    print(x)
```

Why does execution differ? The if statements are separate and both true, so both evaluate. The if/elif pair are connected, so only one evaluates.