

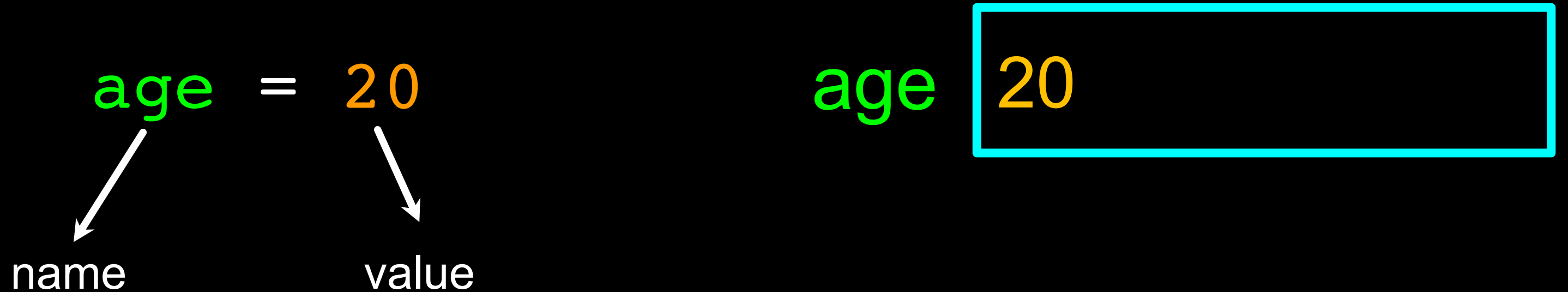
Data Types and `if` Structure

Lecture 2

bingzhi.li@yahoo.com

Variables & Assignment

A **variable** is a memory location used to store a value (**20**)



Python Variable Name Rules

- Must start with a letter or underscore _
- Must consist of letters, numbers, and underscores
- Case Sensitive

Good:

age

age_anne

age23

Bad:

#age

age anne

age.23

Assignment

x	=	2	←	Constant
y	=	$3 * x + (1 - x)$	←	Expression
z	=	<code>input()</code>	←	Function

? Type of x, y, z ?

What Does “Type” Mean?

- In Python variables, literals, and constants have a “**type**”
- Python knows the **difference** between an integer number and a string
- For example “**+**” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'Lucy'
>>> print(eee)
hello Lucy
```

concatenate = put together

Type Matters

- Python knows what “**type**” everything is
- You cannot “add 1” to a string
- We can ask Python what type something is by using the **type()** function

```
>>> eee = 'hello ' + 'Lucy'
>>> eee = eee + 1
Traceback (most recent call last):
File "<stdin>", line 1, in
<module>TypeError: Can't convert
'int' object to str implicitly
>>> type(eee)
<class 'str'>
>>> type('hello')
<class 'str'>
>>> type(1)
<class 'int'>
>>>
```

1. Numbers

- Numbers have two main types
 - **Integers** are whole numbers:
-14, -2, 0, 1, 100, 401233
 - **Floating Point Numbers** have
decimal parts: -2.5 , 0.0, 98.6, 14.0

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class 'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>>
```

Type Conversions

Built-in functions

- `int()` and `float()`

```
>>> i = 42
>>> type(i)
<class'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class'float'>
>>> type(int(f))
<class'int'>
```


String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an `error` if the string does not contain numeric characters

```
>>> strVal = '123'  
>>> type(strVal)  
<class 'str'>  
>>> intVal = int(strVal)  
>>> type(intVal)
```

```
>>> sv = 'hello bob'  
>>> niv = int(sv)  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
ValueError: invalid literal for int()  
with base 10: 'x'
```

2. Booleans

- Only 2 basic members
 - True
 - False

Logical expressions with logical operators : **and** ,**or** and **not**

```
>>> True and False
False
>>> not (True and False)
True
>>> (not True) and False
False
>>> x = True
>>> y = False
>>> x and y
False
>>> y or (not y)
True
```

Comparison Operators

Python	Meaning
<code>==</code>	equality
<code>!=</code>	inequality
<code><</code>	Less than
<code><=</code>	Less than or Equal
<code>>=</code>	Greater than or Equal
<code>></code>	Greater than

Remember: “=” is used for assignment.

```
>>> 3 == 2 + 2
```

```
False
```

```
>>> 3 != 2 + 2
```

```
True
```

```
>>> x = "cat"
```

```
>>> y = "hat"
```

```
>>> x == y
```

```
False
```

```
>>> print(x)
```

```
'cat'
```

```
>>> len(x) > 7 and
```

```
y.count("a") < 3
```

```
False
```

Type

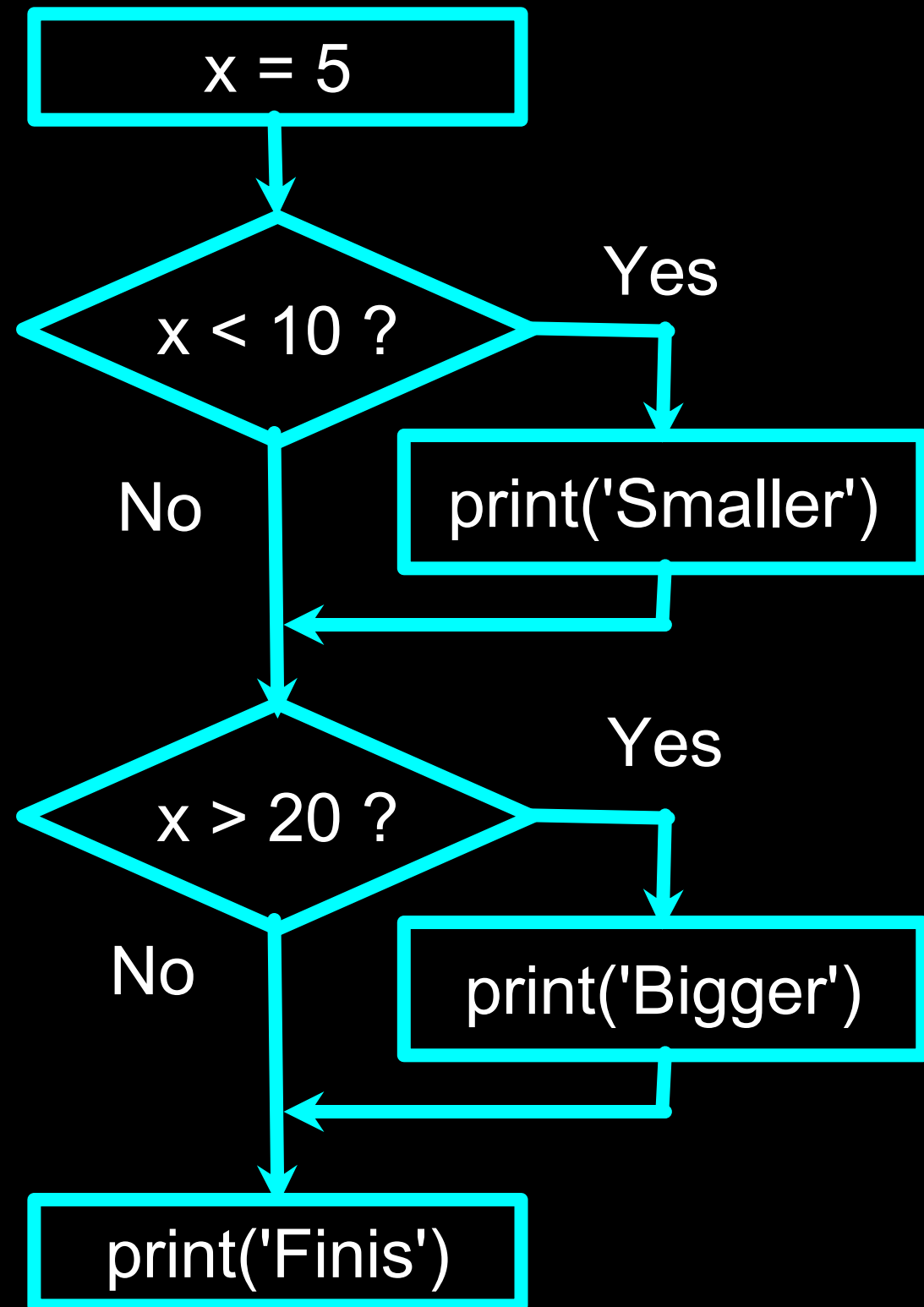
Boolean version of `type()`

- `isinstance()`

```
>>> i = 42
>>> type(i)
<class 'int'>
>>> isinstance(i, int)
True
>>> isinstance(str(i), int)
False
>>> isinstance("hat", str)
True
>>> isinstance(3==4, str)
False
```

Conditional Execution

Conditional Steps



Program:

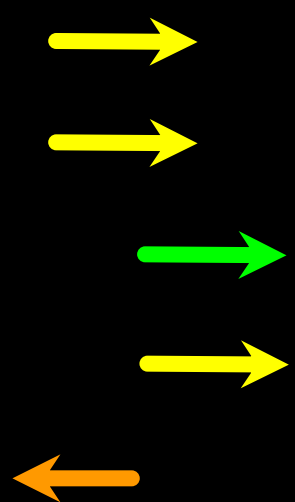
```
x = 5
if x < 10:
    print('Smaller')
if x > 20:
    print('Bigger')

print('Finis')
```

Output:

Smaller
Finis

Indentation



```
x = 5
if x < 10:
    print('Smaller')
    x = x + 1
if x > 20:
    print('Bigger')

print(x)
```

- **increase** after if statement
- **maintain** to indicate the scope
- **reduce** to indicate end of block

Tab key



Indentation

Program:

```
x = 5
if x < 10:
    print('Smaller')
    x = x + 1
if x > 20:
    print('Bigger')

print(x)
```

Must be spaced
tabbed



Output:

Smaller
6

Indentation errors

Program:

```
x = 5
if x < 10:
print( 'Smaller' )
```

Output:

```
File "<stdin>", line 3
    print( 'Smaller' )
    ^
```

IndentationError: expected
an indented block

Indentation errors

Program:

```
x = 5
if x < 10:
    print( 'Smaller' )
    print( 'Bigger' )
```

Output:

```
File "<stdin>", line 4
    print( "Bigger" )
IndentationError: unexpected
indent
```

Indentation errors

- Python cares a *lot* about how far a line is indented. If you mix **tabs** and **spaces**, you may get “**indentation errors**” even if everything looks fine

Program:

```
x = 5
if x < 10:
    print('Smaller')
    print('Bigger')
```

Output:

```
File "<stdin>", line 3
    print("Bigger")
IndentationError: unexpected
indent
```

A difference?

Program1:

2-statement block

```
if 1 + 1 == 3:  
    print( 'that shouldn't happen' )  
    print( 'what?' )
```

Output:

Program2:

1-statement block

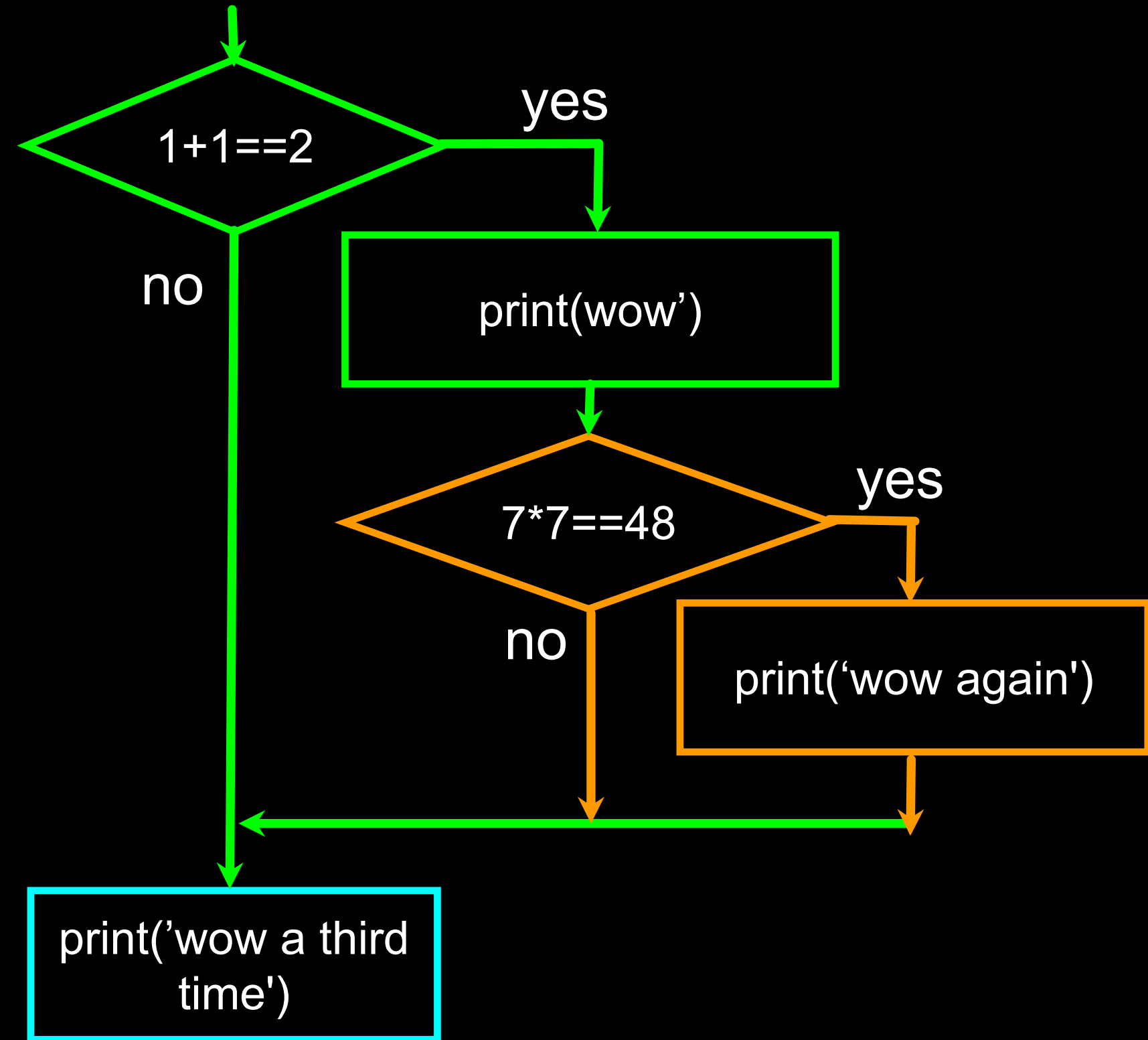
```
if 1 + 1 == 3:  
    print( 'that shouldn't happen' )  
print( 'what?' )
```

Output:
"what?"

Nested Decisions

Program:

```
if 1 + 1 == 2:  
    print('wow')  
    if 7*7 == 48:  
        print('wow again')  
print('wow a third time')
```



Nested Decisions

Program:

```
if 1 + 1 == 2:  
    print('wow')  
    if 7*7 == 48:  
        print('wow again')  
print('wow a third time')
```

Output: ?

'wow'

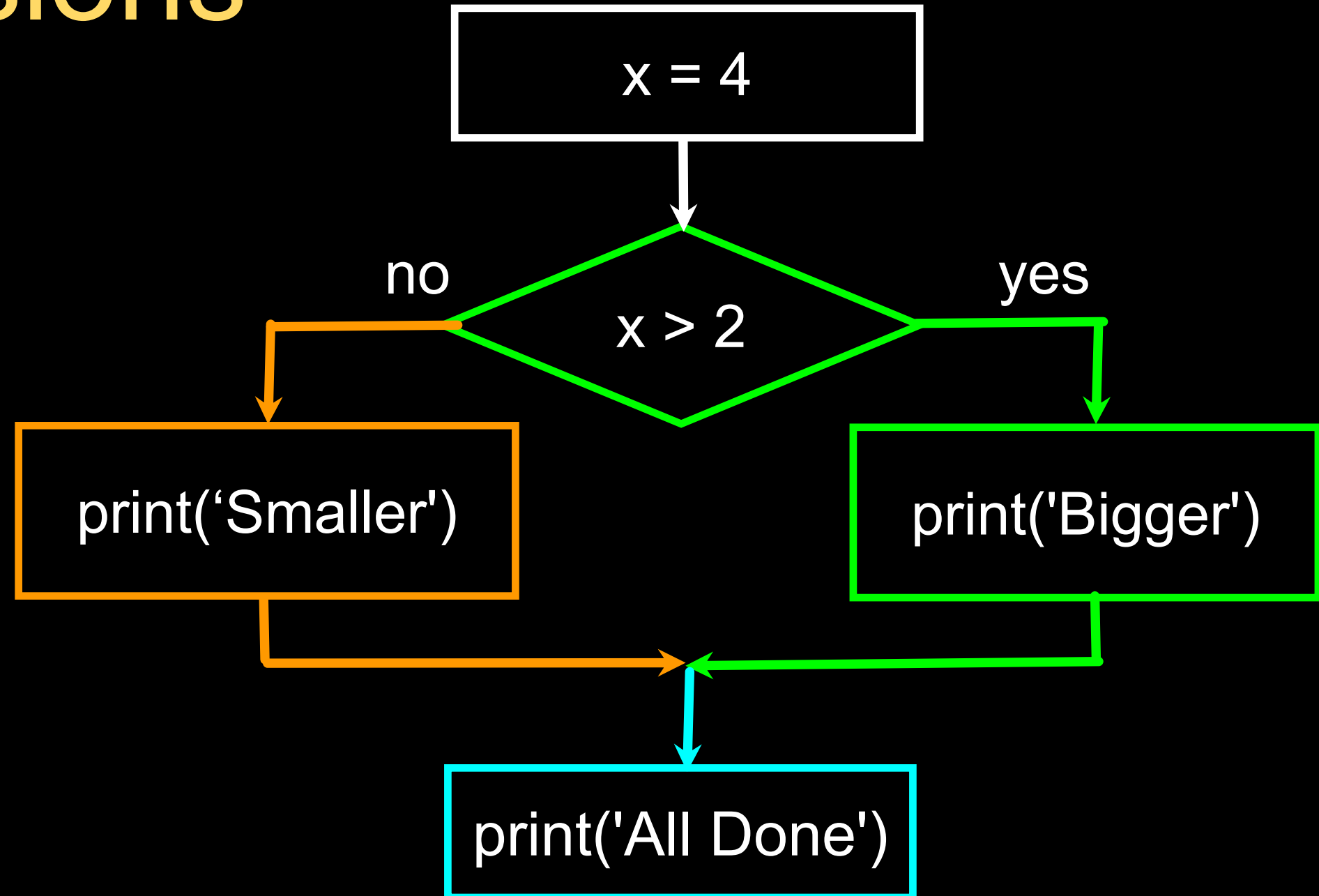
'wow a third time'

Two-way Decisions with `else`:

```
x = 4
```

```
if x > 2 :  
    print('Bigger')  
else :  
    print('Smaller')
```

```
print('All done')
```

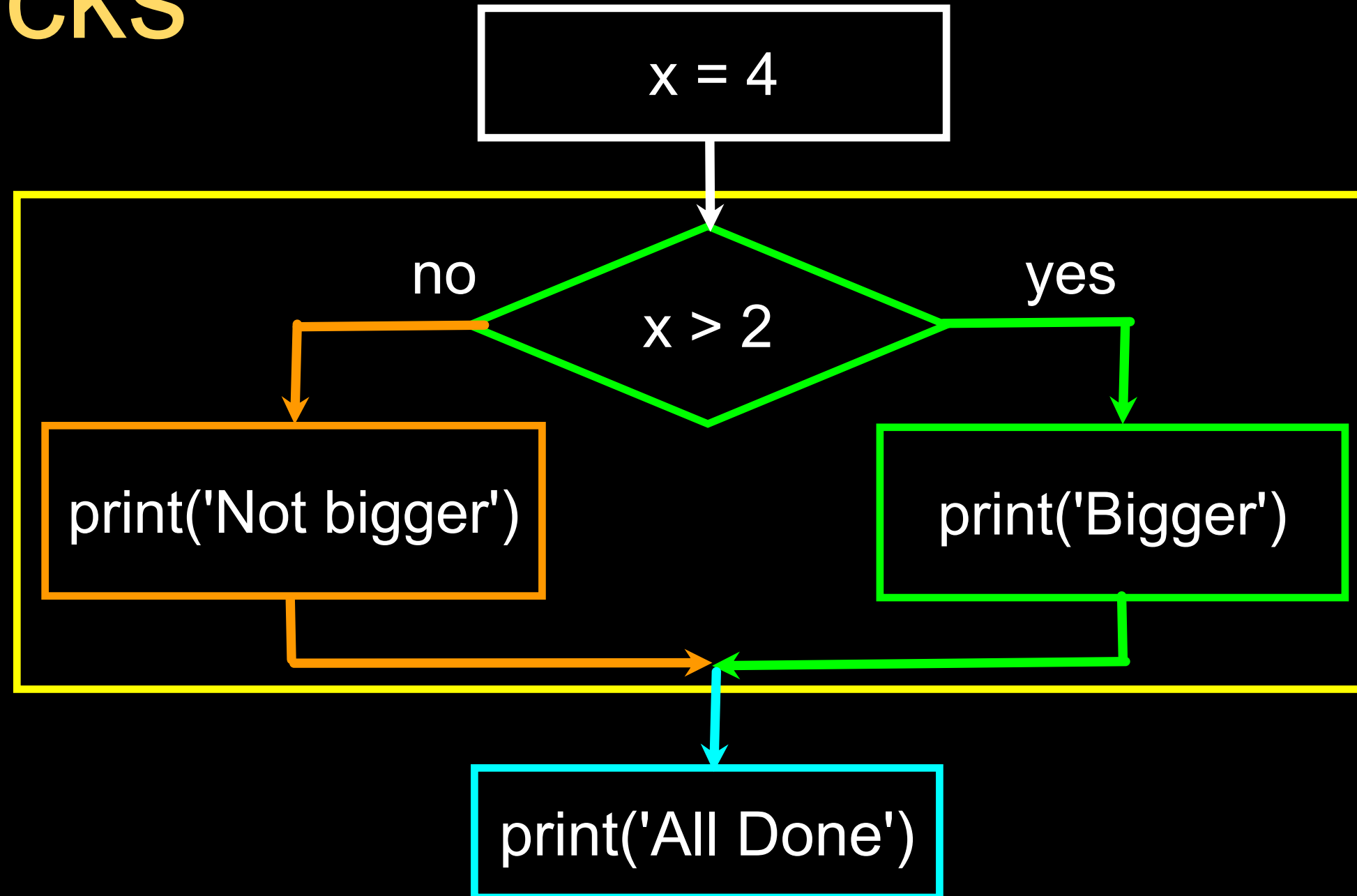


Visualize Blocks

x = 4

```
if x > 2 :  
    print('Bigger')  
else :  
    print('Smaller')
```

```
print('All done')
```



if ... else

If the `if` clause evaluates to `true`, the `else` block will not execute:

```
if 2+2==4:
    print('this will print')
else:
    print("but this won't")
    print('...and neither will this')
```

? output: 'this will print'

if ... else

If the `if` clause evaluates to `true`, the `else` block will not execute:

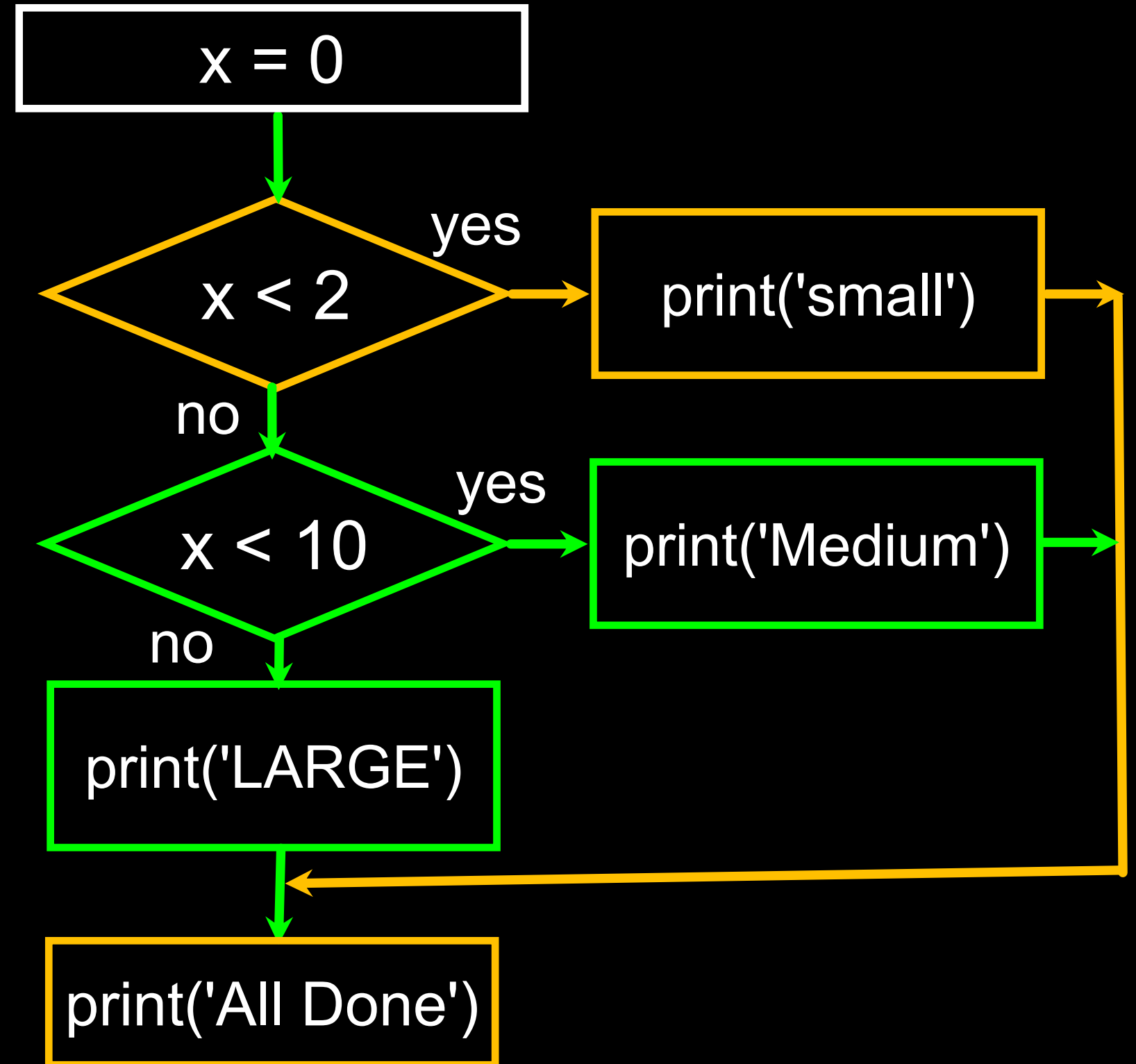
```
if 2+2==10:
    print('this will print')
else:
    print("but this won't")
    print('...and neither will this')
```

? output: "but this won't"
"...and neither will this"

Multi-way

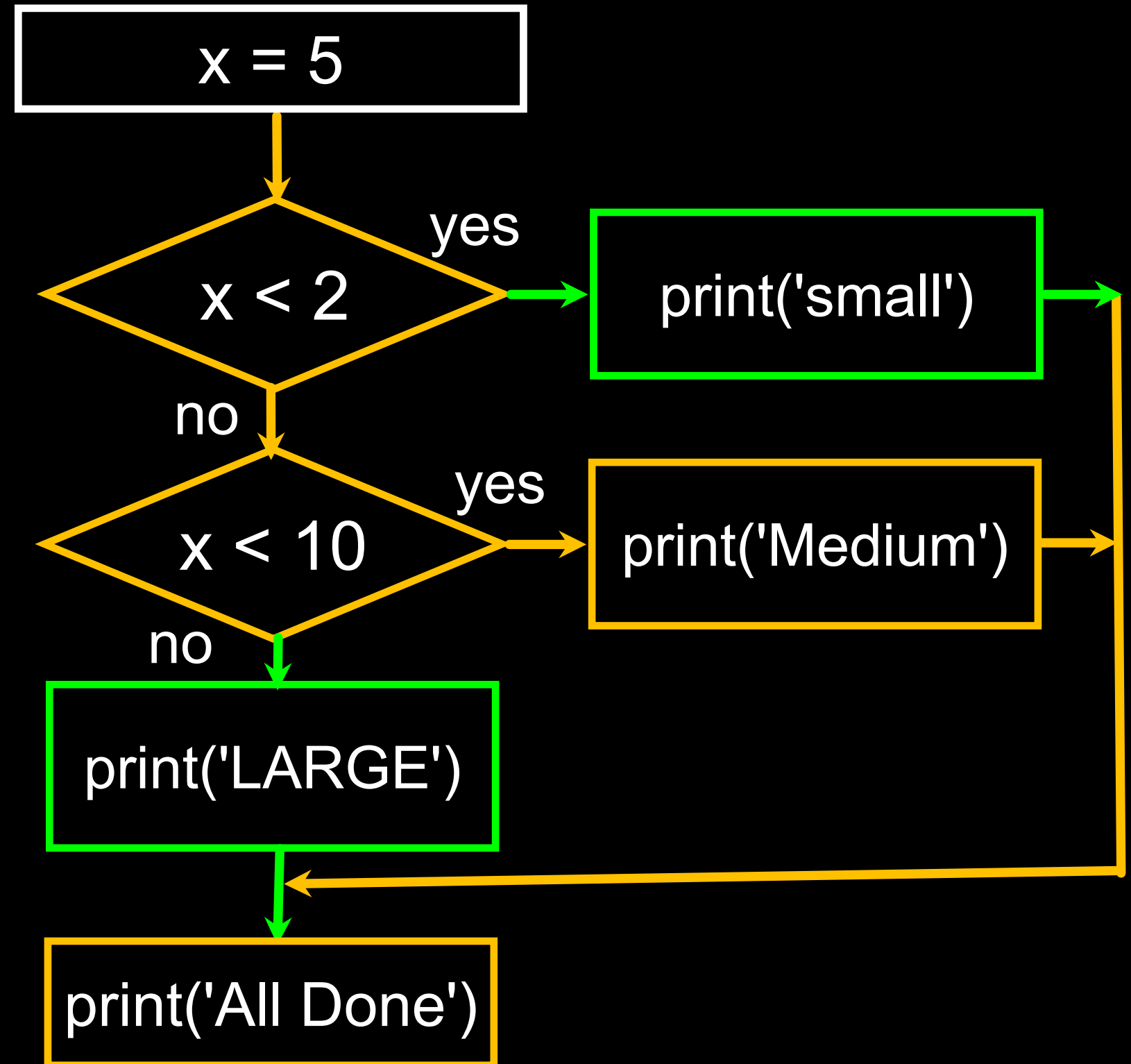
`if ... elif ... else`

```
x = 0
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```



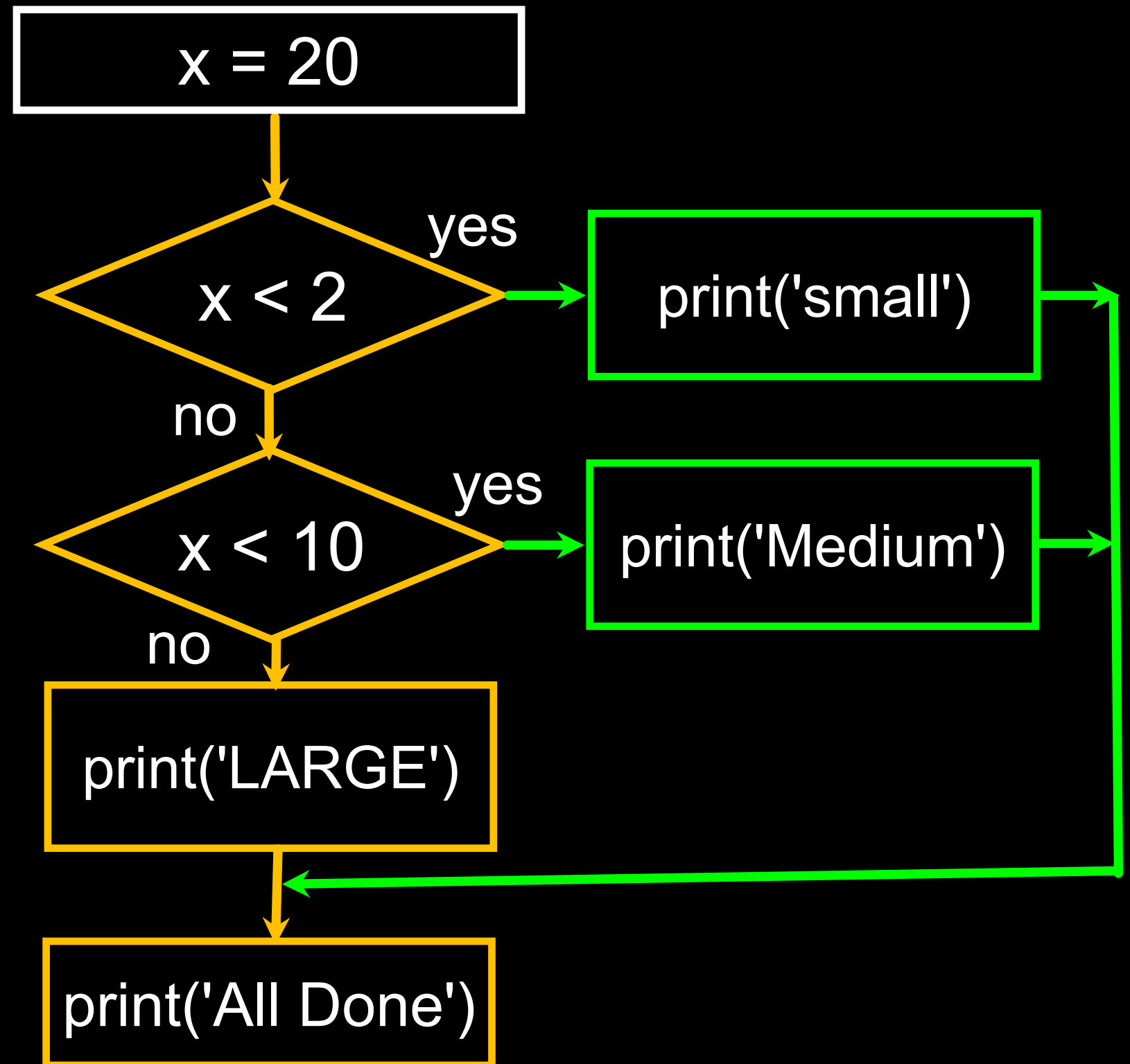
Multi-way

```
x = 5
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```



Multi-way

```
x = 20
if x < 2 :
    print('small')
elif x < 10 :
    print('Medium')
else :
    print('LARGE')
print('All done')
```



Multi-way Puzzles

```
if test1:  
    block1  
elif test2:  
    block2  
elif test3:  
    block3  
else:  
    block4
```

test1	test2	test3	What applies?
True	True	True	block1
True	True	False	block1
True	False	False	block1
False	True	True	block2
False	False	True	block3
False	False	False	block4

User Input

- We can instruct Python to pause and read data from the user using the `input()` function
- The `input()` function returns a string

```
>>>nam = input('Type your name: ')\n>>>print('Welcome', nam)
```

```
Type your name: Lucy\n'Welcome Lucy'
```


Converting User Input



- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function
- Later we will deal with bad input data

```
inp = input('Europe floor?')  
usf = int(inp) + 1  
print('US floor', usf)
```

Europe floor? 0
US floor 1