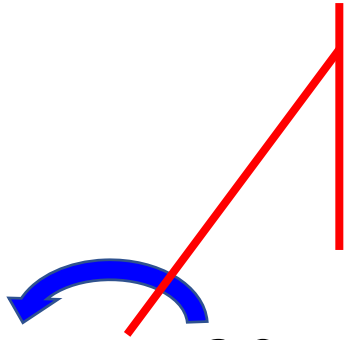
Syntax: conditionals
Summary

pythontutor.com

Variables

```
hours = 30
rate = 12.50
```

Assignment statement



NOT mathematical operator!

```
hours = 30
rate = 12.50
rate = 1.5 * rate
pay = hours * rate
print(pay)
```

Numeric expressions

First evaluate then assign

```
norm_hours = 40
over_hours = 5
rate = 12.50
full_pay = norm_hours * rate +
           over_hours * rate *1.5

print(full_pay)
```

String expressions

```
happy = 'happy birthday to you\n'
name  = 'Marina'
song = happy*2 +
       'Happy birthday, dear ' +
       name + '\n' + happy
print (song)
```

Adding unpredictability

```
import random

rand_num = random.randint (1,15)
rand_selection =
  random.choice(['rock’,'paper','scissors’])
rand_selection = random.choice([10, 20, 30])
```

Input/output

Pauses, waits, then assigns

```
name = input('Who are you? ')
print('Welcome', name)
```

Branching programs

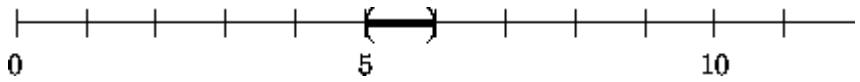
- *Boolean expressions* ask a question and produce a Yes or No result which we use to control program flow
- Boolean expressions using comparison operators evaluate to True / False

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

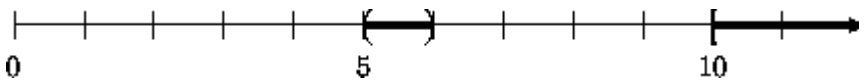
Boolean (logical) operators: and, or, not

- “Cats have whiskers **and** dogs have tails“
- “Cats have whiskers **and** dogs have wings“
- “Cats have whiskers **or** dogs have tails“
- “Cats have whiskers **or** dogs have wings“
- **not** “Cats have whiskers”
- **not** “Dogs have wings“

Boolean expressions with numbers

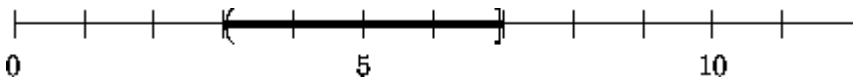


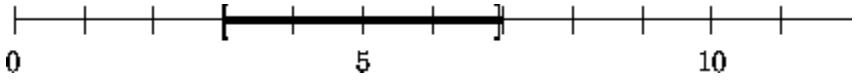
$x > 5$ and $x < 6$

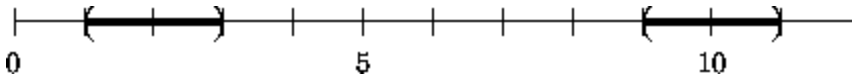


($x > 5$ and $x < 6$) or $x >= 10$

Translate intervals for *x* into code:





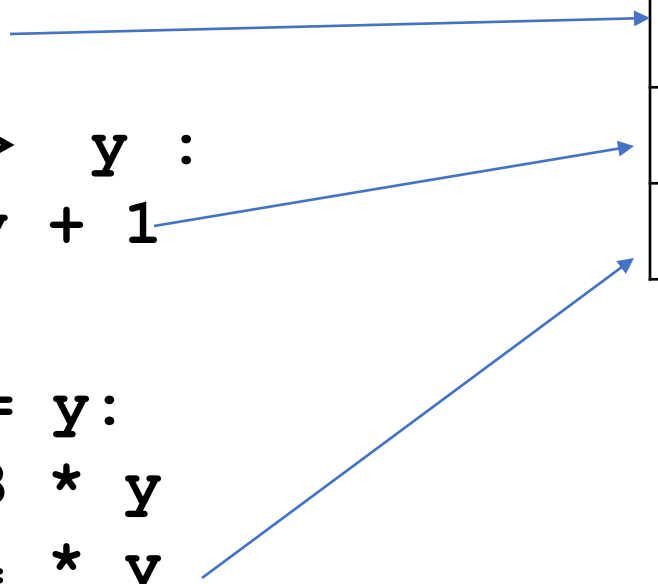


Trace code by hand!

```
x = 5
y = 4
if x  >  y :
  y = y + 1


if x == y:
  y = 3 * y
  x = 4 * y
else:
  y = 2*y
  x = 4*y
print (x, y)
```

x	y
5	4
5	5
60	15



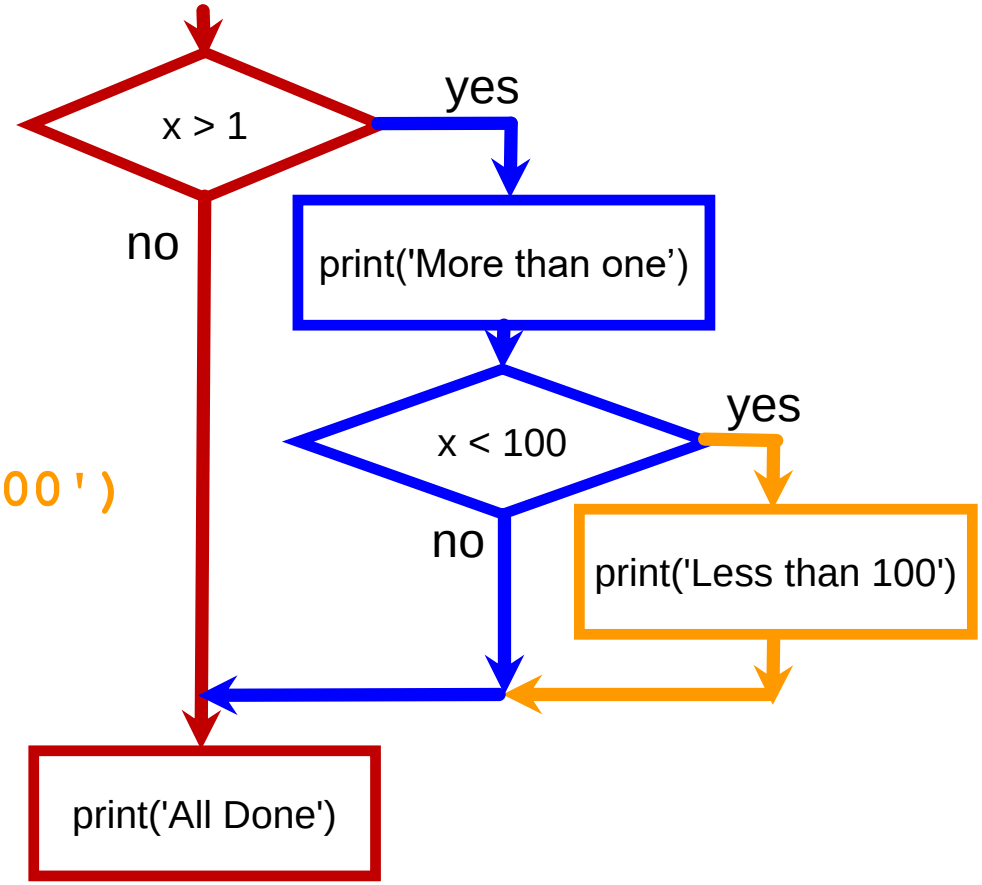
Program is built from blocks (paragraphs)

```
x = 5
if x > 2 :
    print('Bigger than 2')
    print('Still bigger')
print('Done with 2')

i = 4
if i < 100:
    print(i)
    if i > 2 :
        print('Bigger than 2')
    print('Done with i', i)
print('All Done')
```

Nested Conditionals

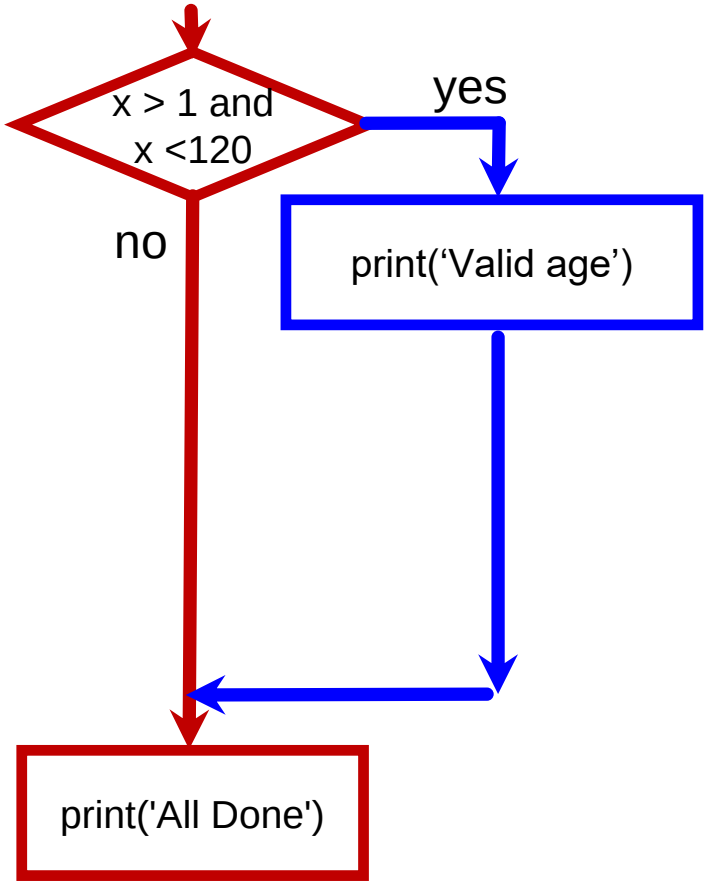
```
if x > 1 :
    print('More than one')
    if x < 100 :
        print('Less than 100')
print('All done')
```



```
import this
```

“Flat is better than nested”

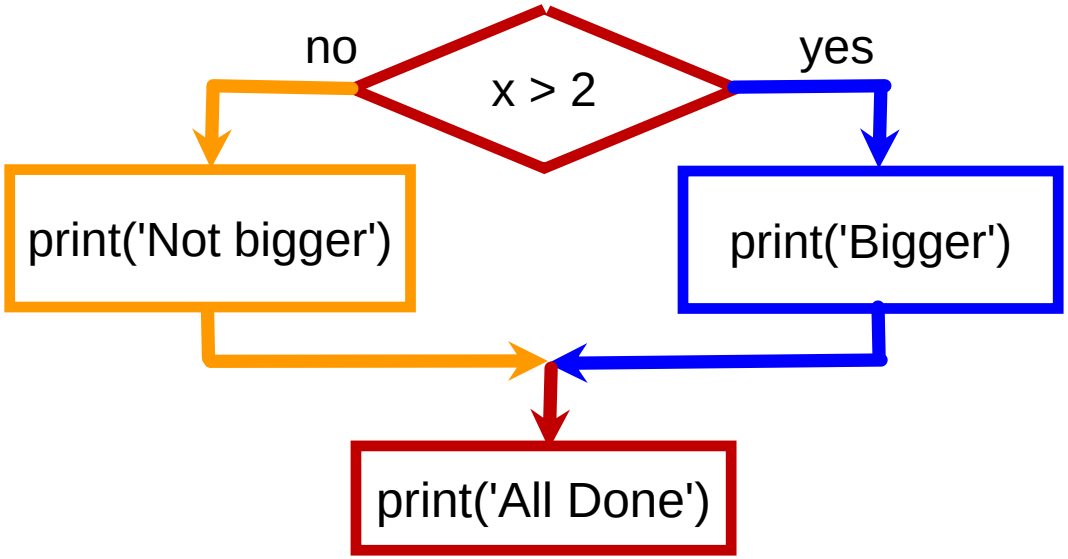
```
if x > 1 and x < 120:
    print('Valid age')
print('Program continues')
```



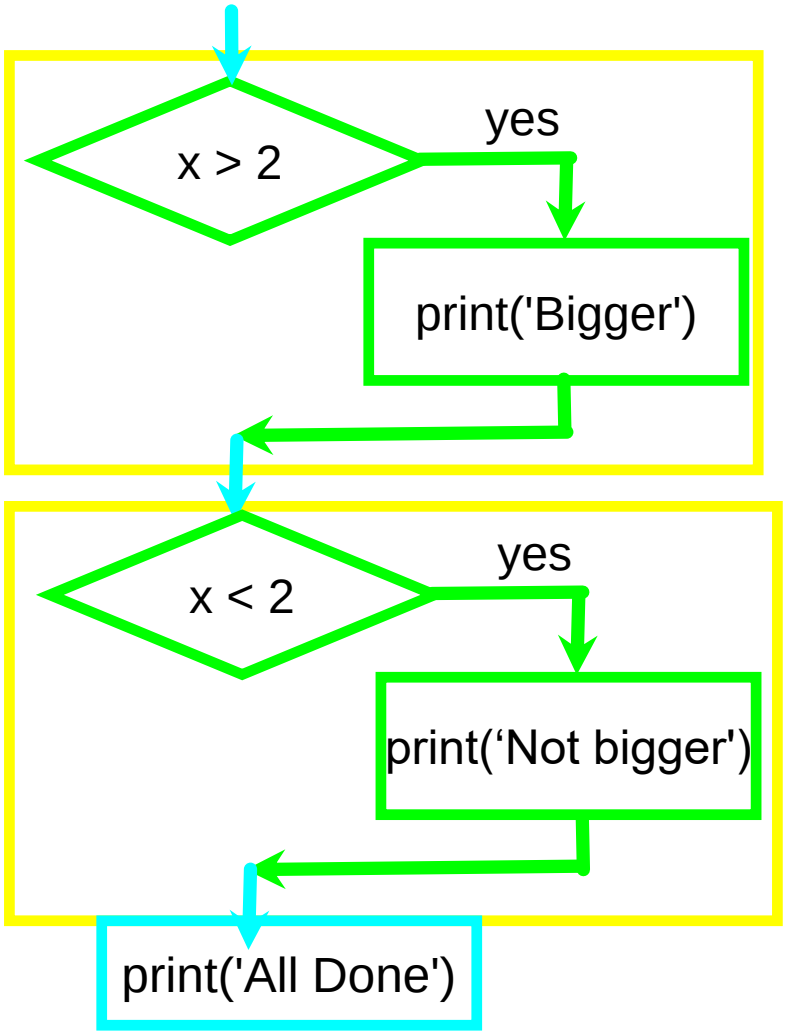
Two-way crossroads: single question

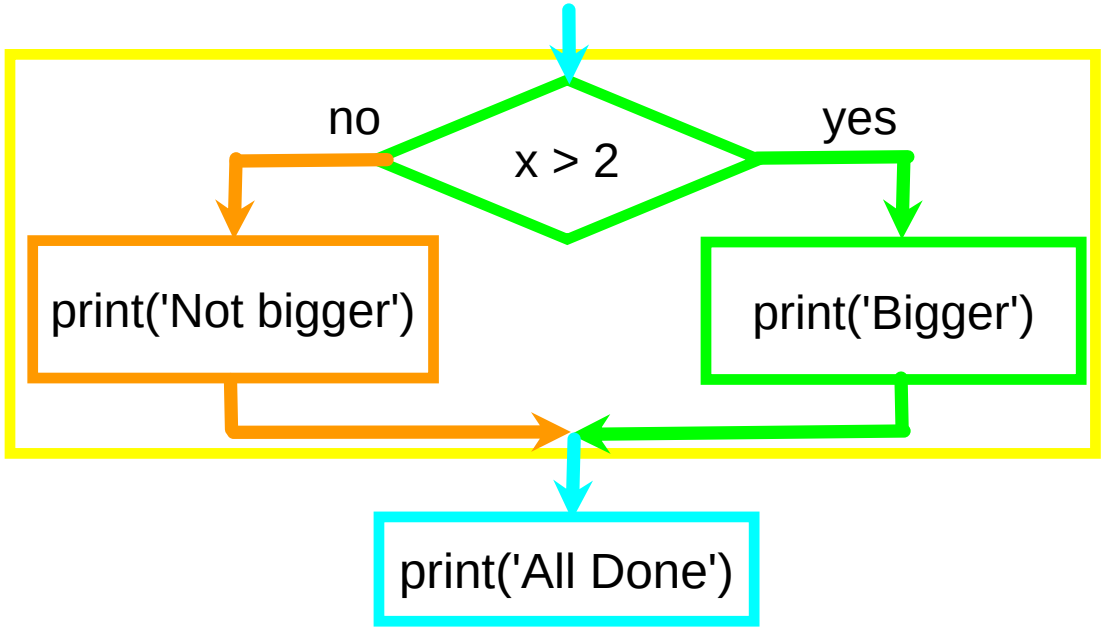
```
if x > 2 :
    print('Bigger')
else :
    print('Smaller')

print('All done')
```



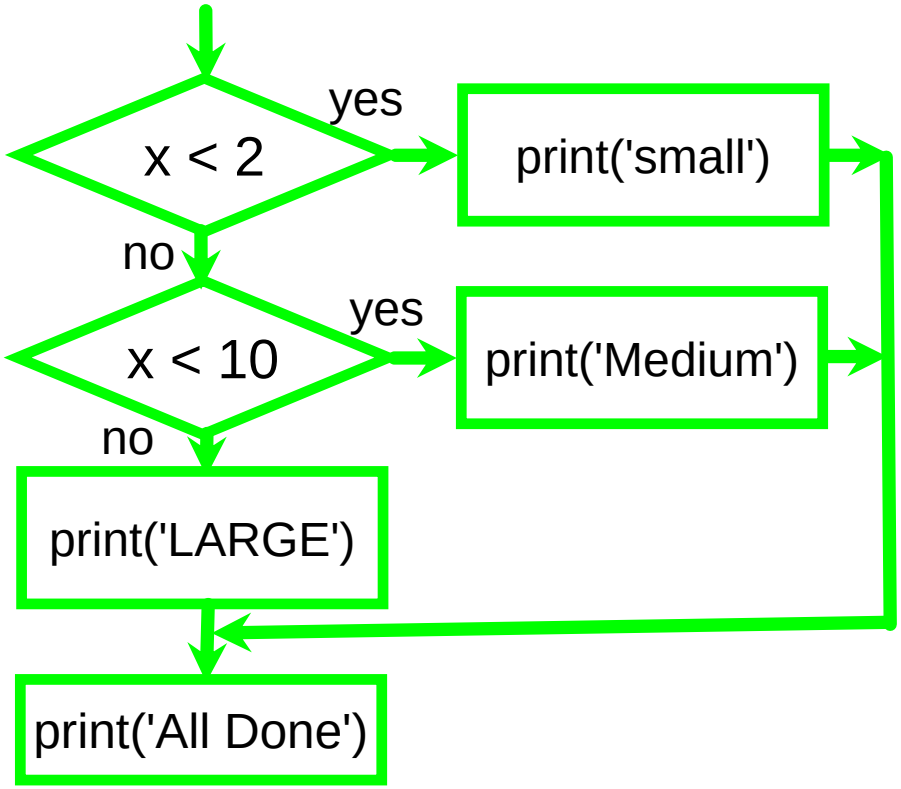
Compare: when the result is different?





Multi-way crossroads

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



Multi-way examples

```
# No Else
x = 1
if x < 2 :
    print('Small')
elif x < 10 :
    print('Medium')

print('All done')
```

Multi-way examples

```
if x < 2 :
    print('Small')
elif x < 10 :
    print('Medium')
elif x < 20 :
    print('Big')
elif x < 40 :
    print('Large')
elif x < 100:
    print('Huge')
else:
    print('Ginormous')
```

Multi-way Puzzle 1

Which message will never be printed regardless of the value for x?

```
if x < 2 :
    print('Below 2')
elif x >= 2 :
    print('Two or more')
else:
    print('Something else')
```

Multi-way Puzzle 2

Which message will never be printed regardless of the value for x?

```
if x < 2 :
    print('Below 2')
elif x < 20 :
    print('Below 20')
elif x < 10 :
    print('Below 10')
else:
    print('Something else')
```

Values and variables have **types**

Numeric types

float

int

bool

```
>>> type(3.14)
<class 'float'>
```

```
>>> type(True)
<class 'bool'>
```

Sequence types

str

list

```
>>> type('writer')
<class 'str'>
```

```
>>> type([1,2,3])
<class 'list'>
```

Type conversions may be dangerous

We can always apply ***str*** to convert number to string
Converting string to number is not always possible!

```
>>> raw_input = input ('enter number: ')
enter number: 25
>>> number = int (raw_input)
>>> print (number)
25
>>>
```

Type conversions may be dangerous

We can always apply *str* to convert number to string
Converting string to number is not always possible!

```
>>> raw_input = input ('enter number: ')
enter number: two
>>> number = int (raw_input)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    number = int (raw_input)
ValueError: invalid literal for int() with base 10: 'two'
```

Surround dangerous code with try - except

```
rawstr = input('Enter a positive number:')
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print('Nice work')
else:
    print('Not a positive number,
                  maybe not a number at all')
```