Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 6 – 17, 2020

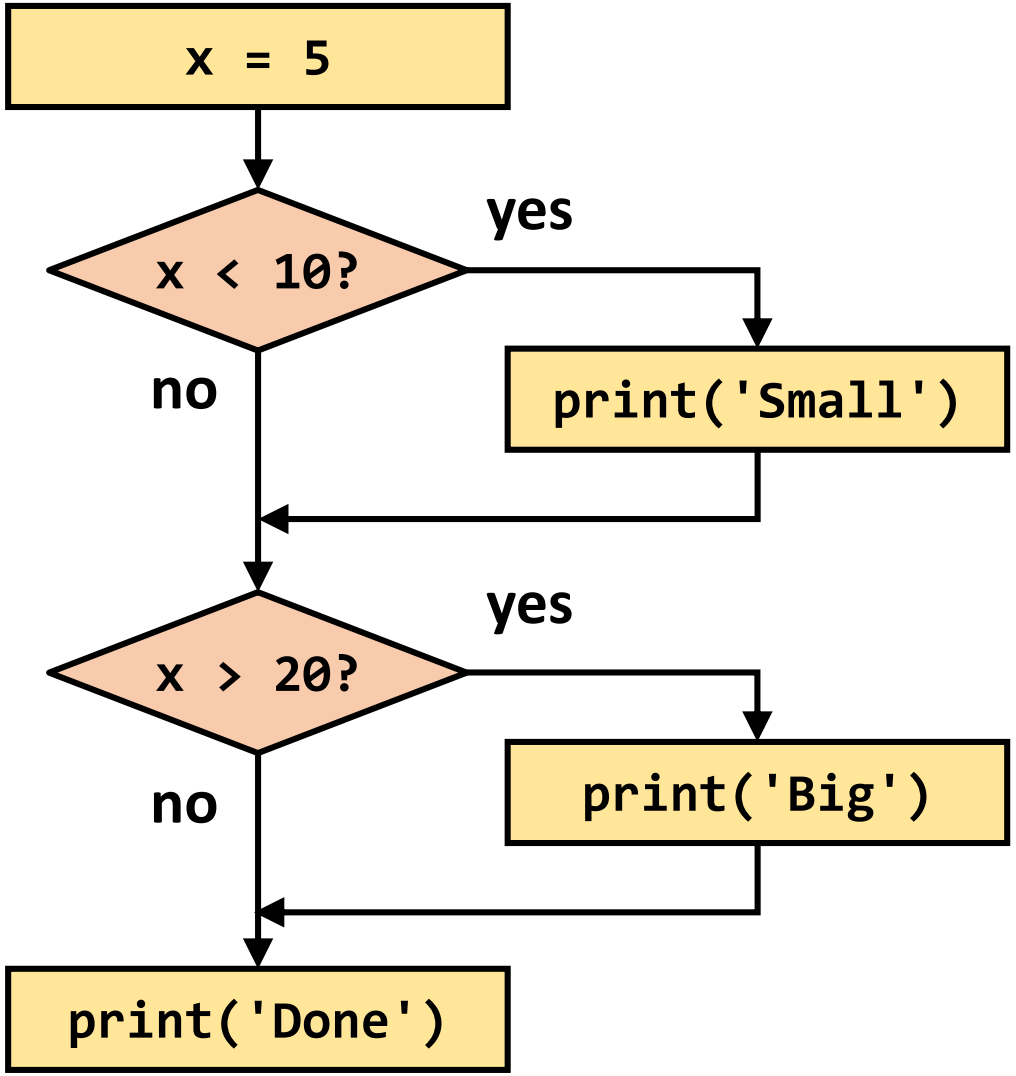
Python for Data Analytics

Control Flow

Conditionals

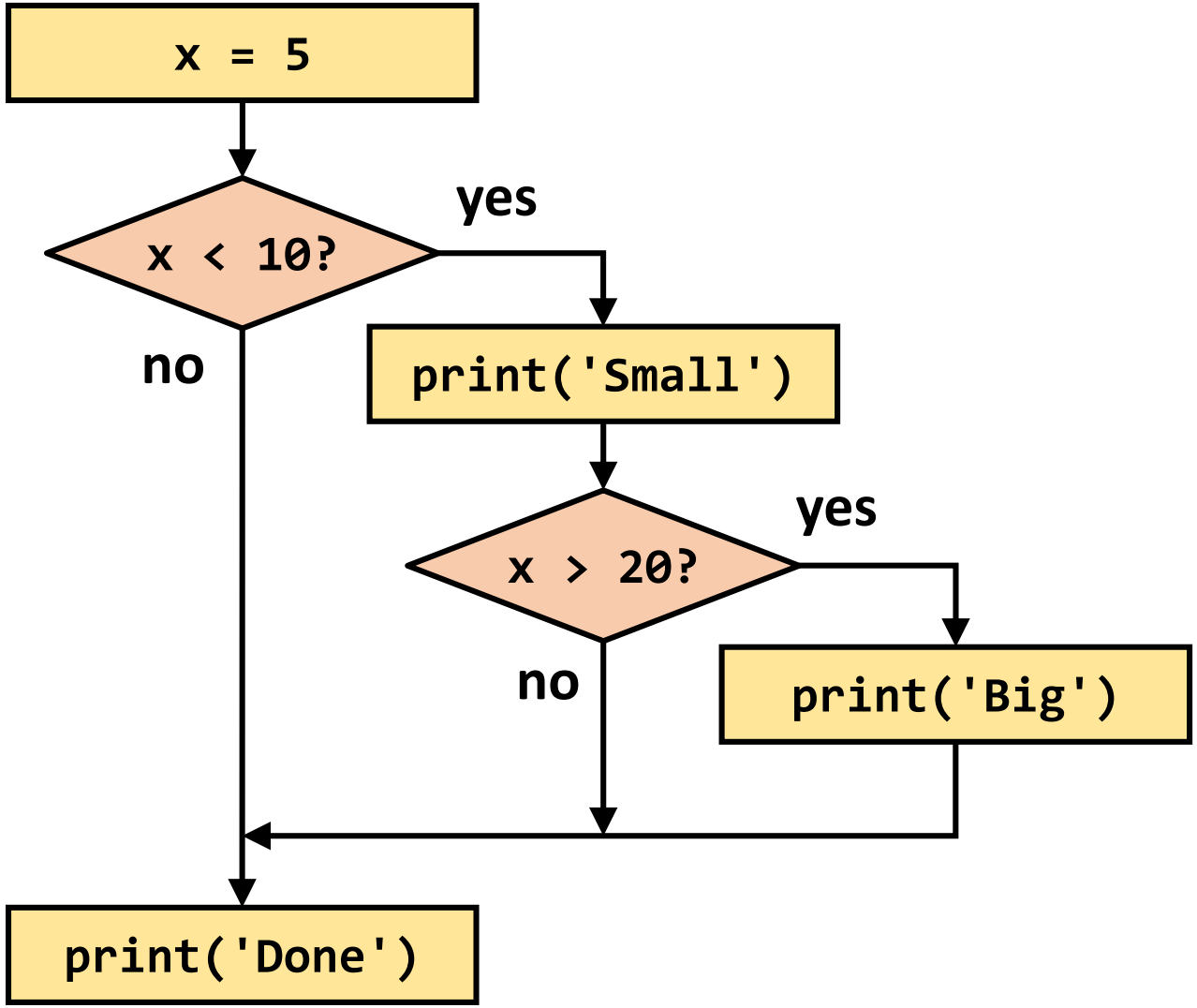
Conditional Steps

```
x = 5
if x < 10:
    print('Small')
if x > 20:
    print('Big')
print('Done')
```



Indentation Matters!

```
x = 5
if x < 10:
    print('Small')
    if x > 20:
        print('Big')
print('Done')
```



Indentation

- Python does not use curly braces { } to indicate a block of statements
- Increase indent after an `if`, `elif`, `else`, `for`, `while` etc. statement
- Maintain indent to indicate the scope of the block
- Reduce indent back to indicate the end of the block
- Blank / comment lines are ignored
- Turn off tabs! (turn tabs into spaces)

```
if a < b:
    a = a + b
    print('here')
else:
    a = a - b
    print('there')
print('done')
```

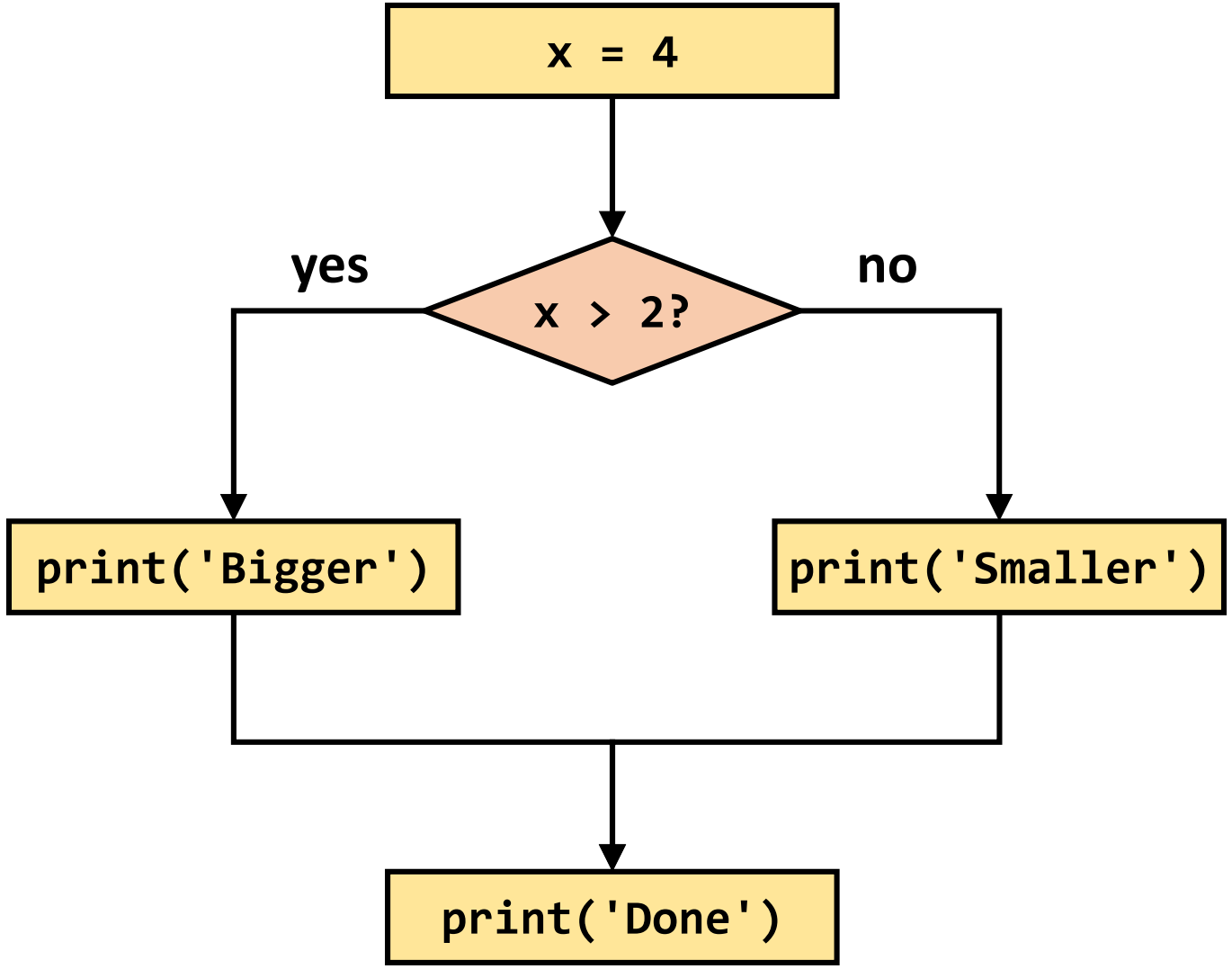
Evaluating Conditions

- Boolean expressions using comparison operators evaluate to `True` or `False`
- Several Boolean expressions can be combined using logical `and` / `or` / `not` operators
- Comparison operators do not change the variables

Notation	Meaning
`a < b`	True if **a** is less than **b**
`a <= b`	True if **a** is less than or equal to **b**
`a == b`	True if **a** is equal to **b**
`a != b`	True if **a** is not equal to **b**
`a >= b`	True if **a** is greater than or equal to **b**
`a > b`	True if **a** is greater than **b**
`A and B`	True if both **A** and **B** are True
`A or B`	True if either **A** or **B** (or both) is True
`not A`	True if **A** is False
`A is B`	True if **A** and **B** point to the same object
`A is not B`	True if A and B do not point to the same object

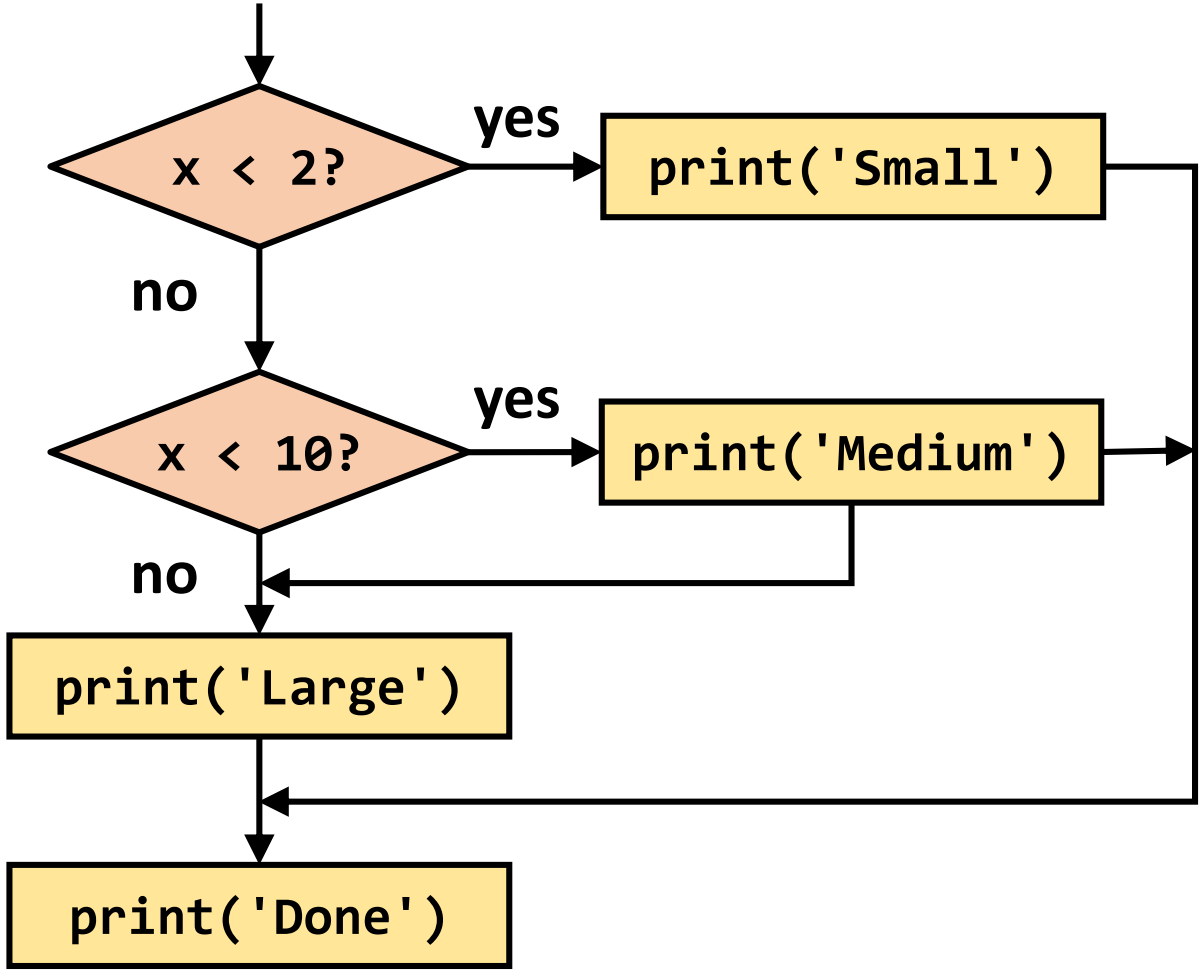
Two-way Decisions

```
x = 4
if x > 2:
    print('Bigger')
else:
    print('Smaller')
print('Done')
```



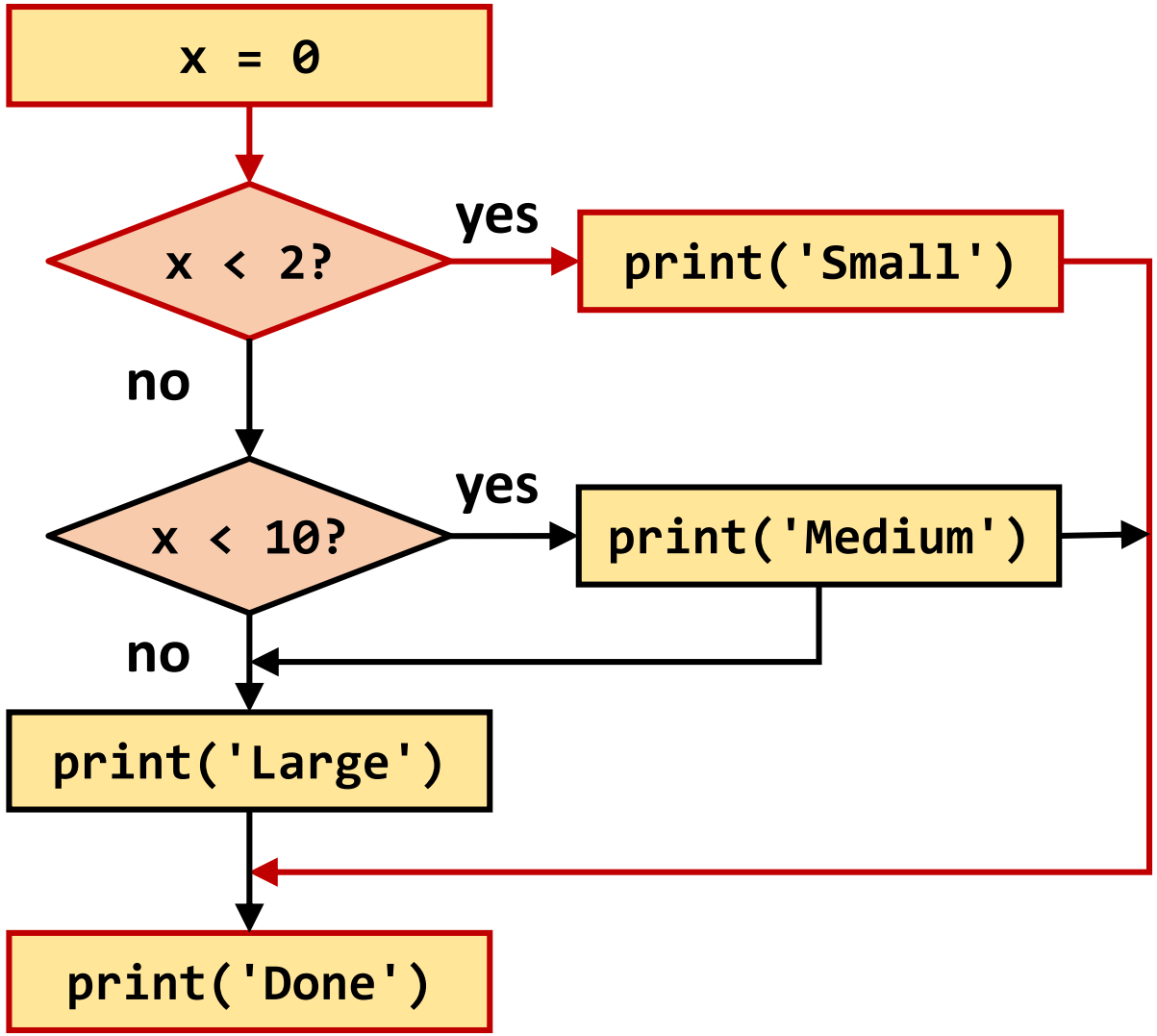
Multi-way Decisions (1)

```
if x < 2:
    print('Small')
elif x < 10:
    print('Medium')
else:
    print('Large')
print('Done')
```



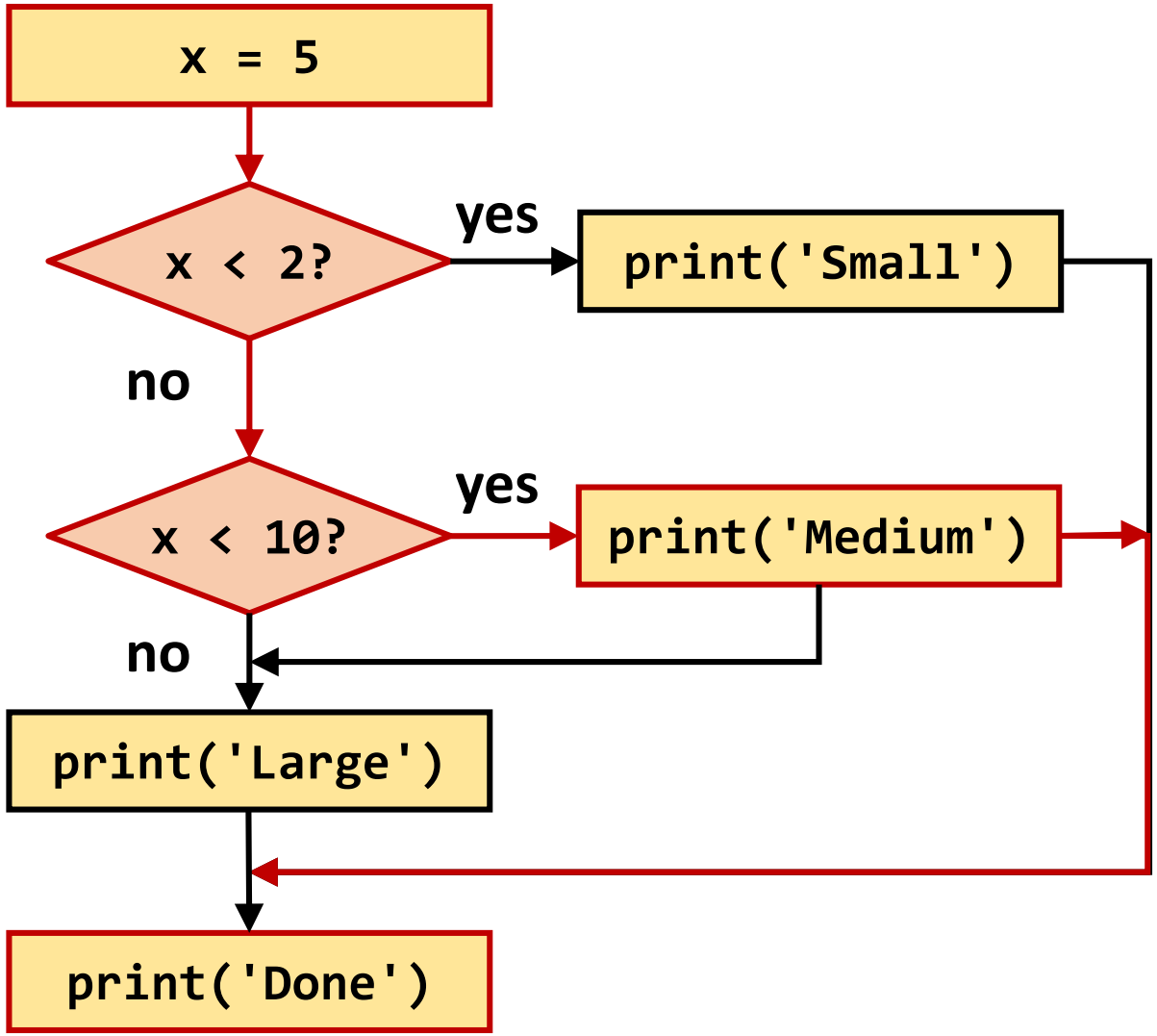
Multi-way Decisions (2)

```
x = 0
if x < 2:
    print('Small')
elif x < 10:
    print('Medium')
else:
    print('Large')
print('Done')
```



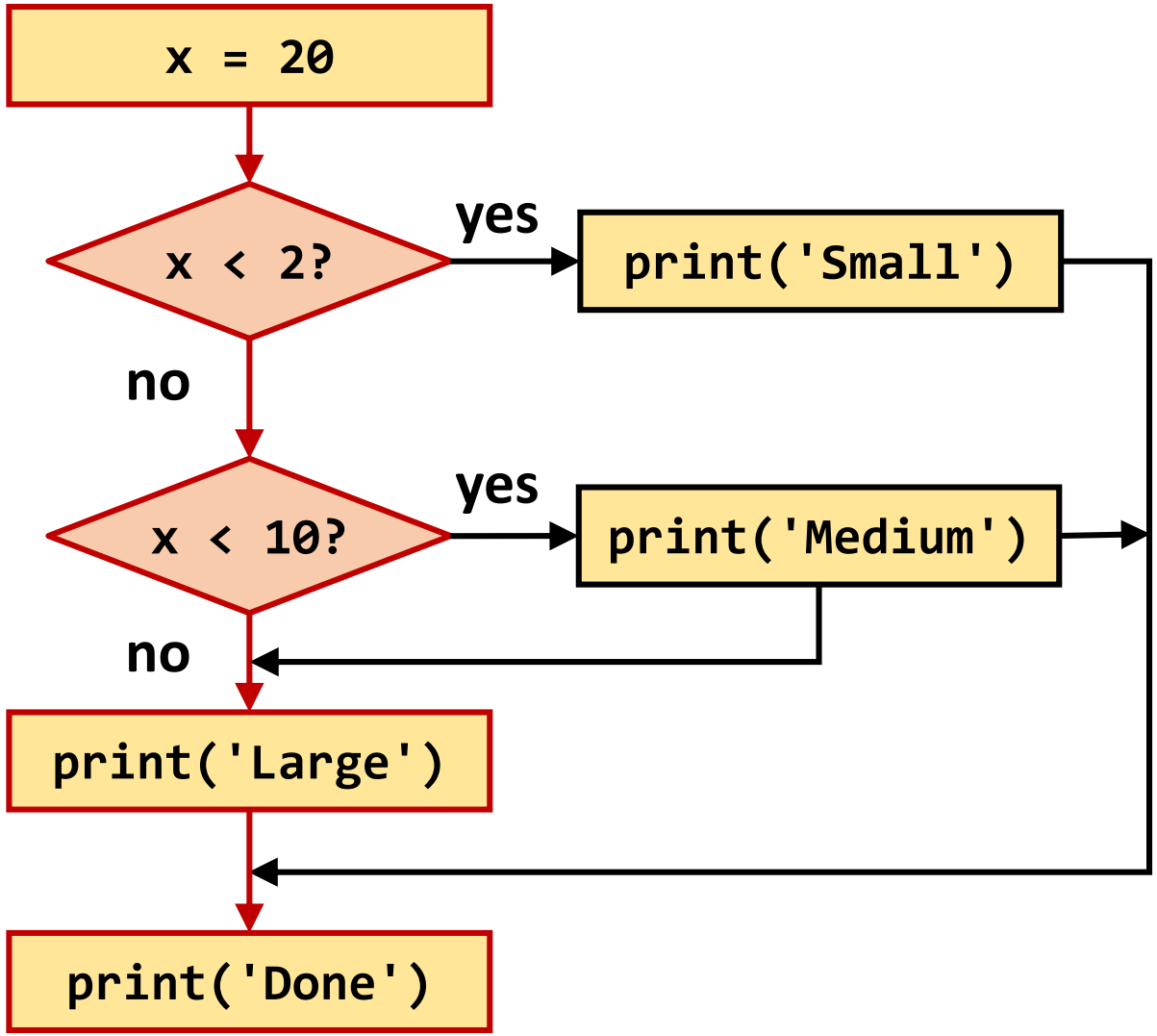
Multi-way Decisions (3)

```
x = 5
if x < 2:
    print('Small')
elif x < 10:
    print('Medium')
else:
    print('Large')
print('Done')
```



Multi-way Decisions (4)

```
x = 20
if x < 2:
    print('Small')
elif x < 10:
    print('Medium')
else:
    print('Large')
print('Done')
```



Multi-way Puzzles

- What's wrong with these programs?

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

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

Conditional Expression

```
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
elif score >= 60:
    grade = 'D'
else:
    grade = 'F'
```

```
grade = 'A' if score >= 90 else \
        'B' if score >= 80 else \
        'C' if score >= 70 else \
        'D' if score >= 60 else \
        'F'
```

Loops

Loops

- `while` loop
  - Keep running the loop body while expression is `True`

```
while (expression):
    <statement_1>
    <statement_2>
        ...
    <statement_n>
```

- `for` loop
  - Run the loop body for the specified range

```
for <element> in <object>:
    <statement_1>
    <statement_2>
        ...
    <statement_n>
```

Loops Example

- `while` loop
  - Keep running the loop body while expression is `True`

```
i = 0
while i < 5:
    print(i)
    i += 1
```

- `for` loop
  - Run the loop body for the specified range

```
for i in range(5):
    print(i)
```

While vs. For

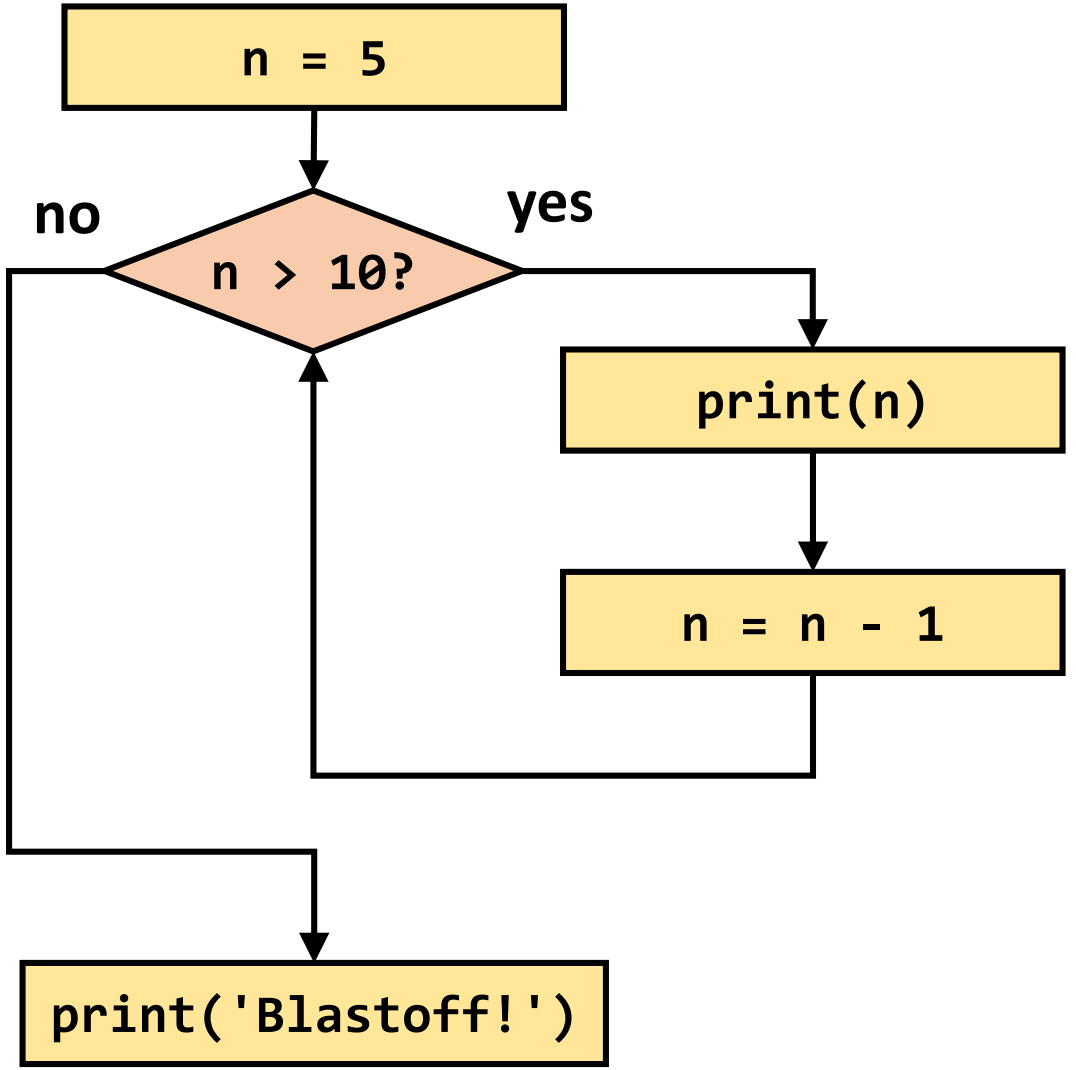
- Indefinite loop – `while`
  - `while` is natural to loop an indeterminate number of times until a logical condition becomes False

- Definite loop – `for`
  - `for` is natural to loop through a list, characters in a string, etc. (anything of determinate size)
  - Rune the loop once for each of the items

Indefinite Loop with `while`

- Indefinite loops have iteration variables that change each time through a loop

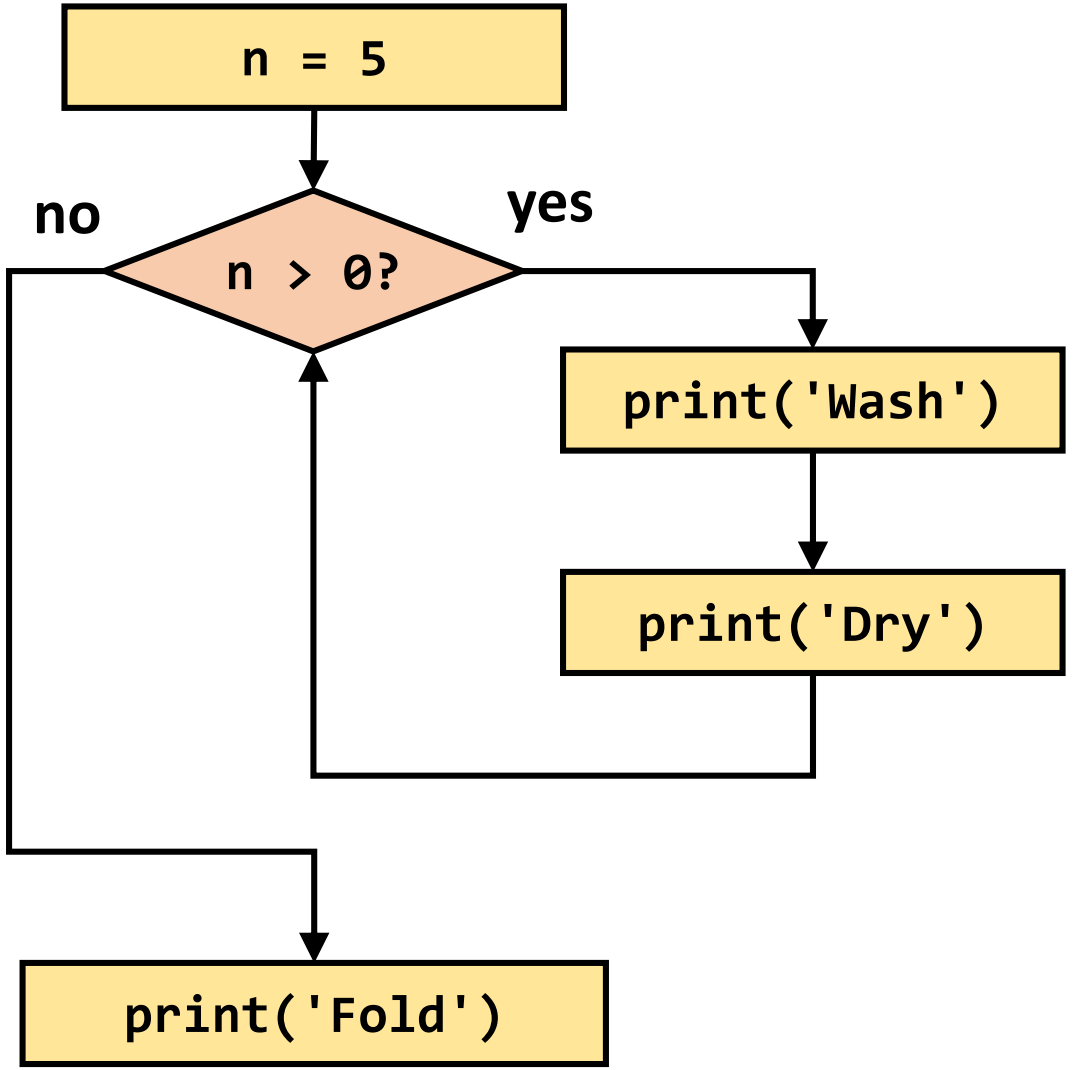
```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Blastoff!')
```



An Infinite Loop

- What's wrong with this loop?

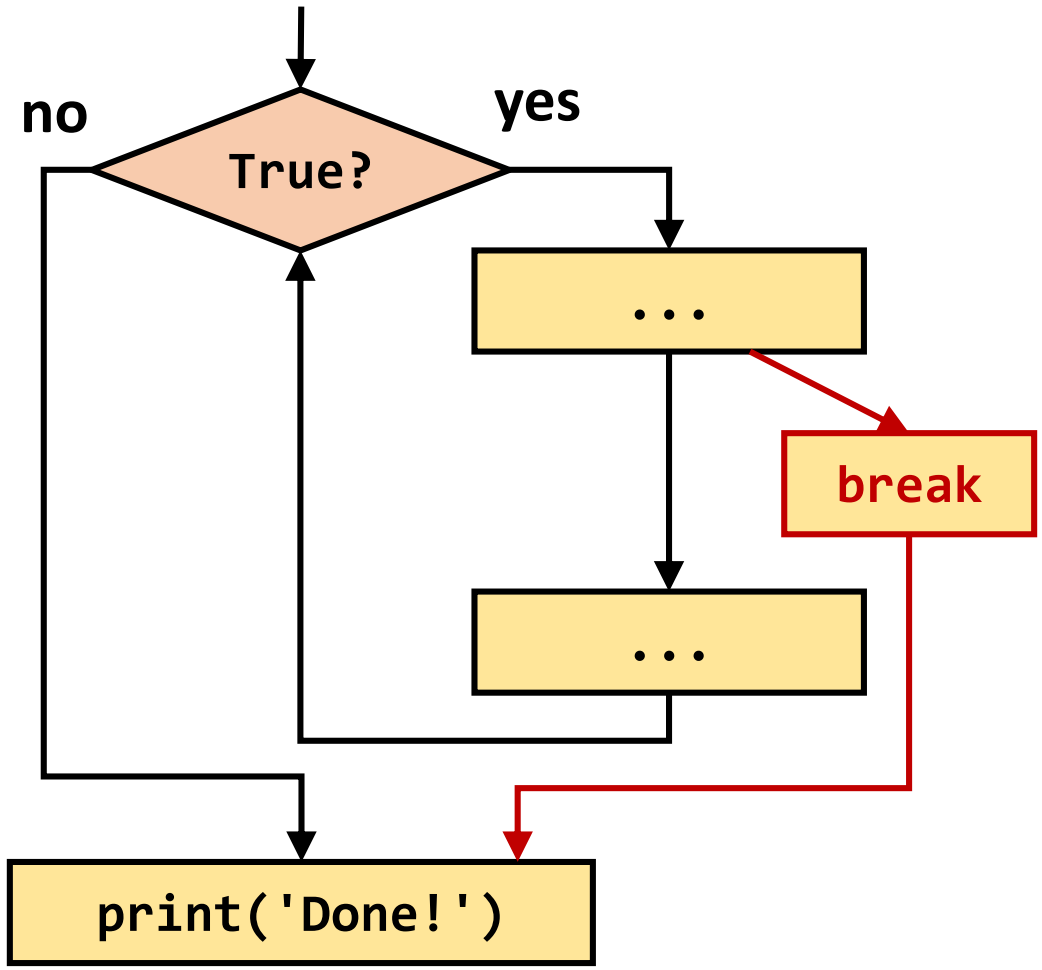
```
n = 5
while n > 0:
    print('Wash')
    print('Dry')
print('Fold')
```



break: Breaking Out of a Loop

- The `break` statement ends the current loop and jumps to the statement immediately following the loop

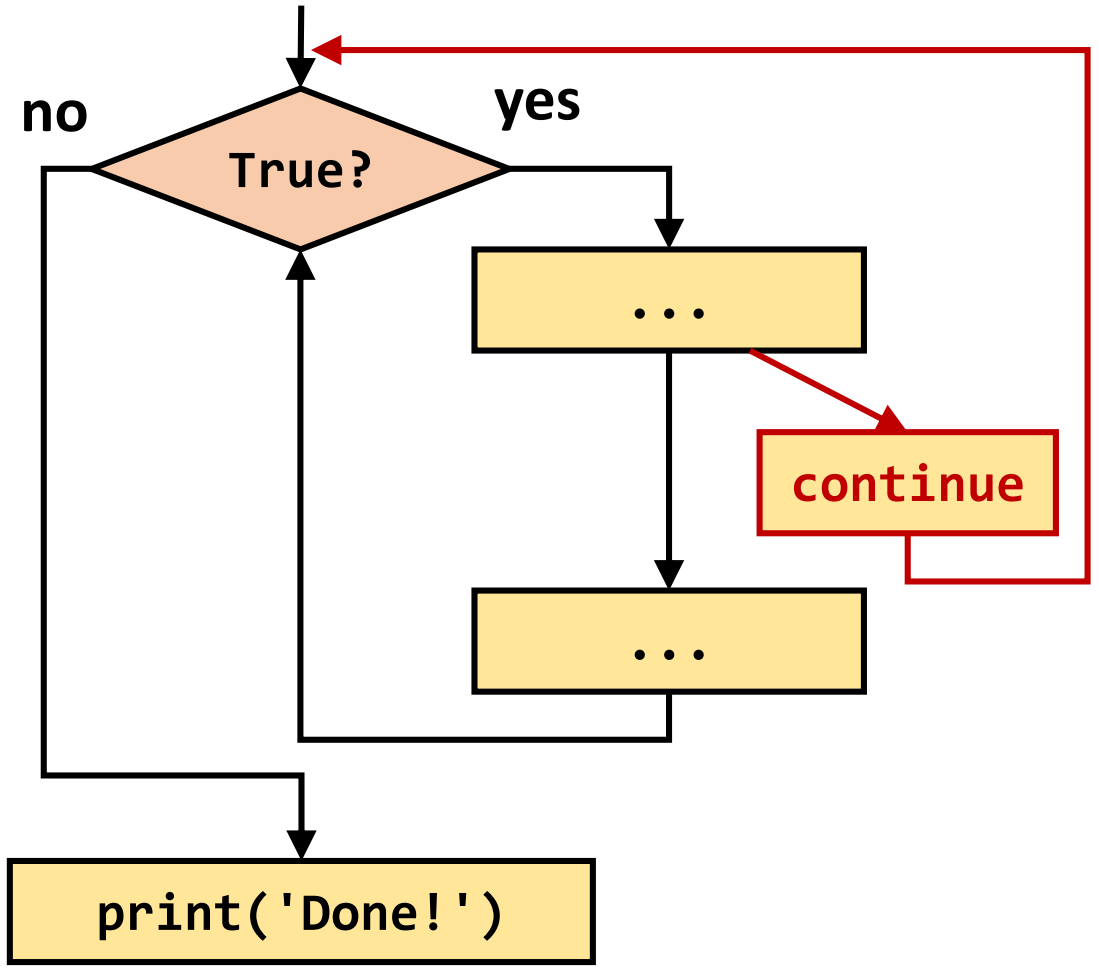
```
while True:
    line = input('> ')
    if line == 'done':
        break
    print(line)
print('Done!')
```



continue: Finishing an Iteration

- The `continue` statement ends the current iteration and jumps to the top of the loop and starts the next iteration

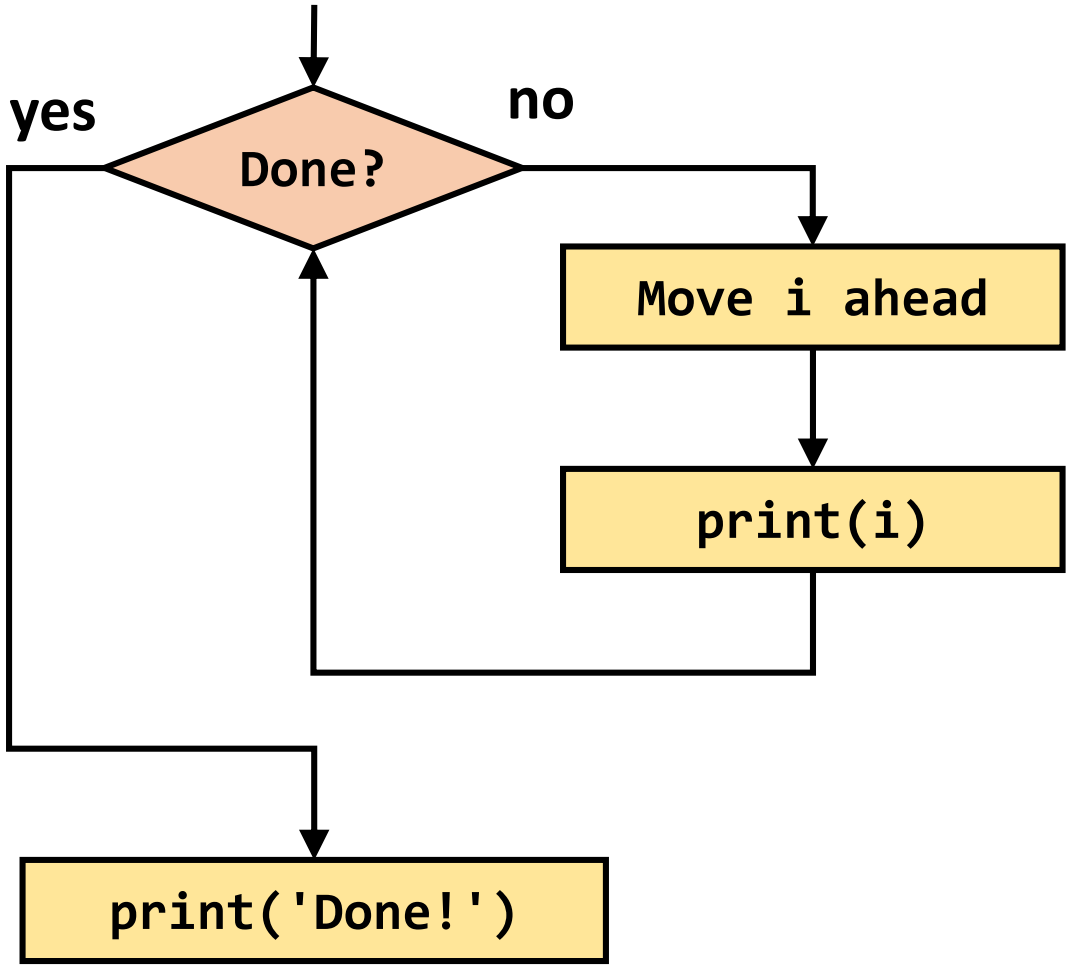
```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print(line)
print('Done!')
```



Definite Loop with for

- Definite loops have explicit iteration variables that change each time through a loop

```
for i in range(5):
    print(i)
print('Done!')
```



Specifying an Integer Range

- `range(start, stop[, step])`
  - Represents an immutable sequence of numbers
  - If the `step` argument is omitted, it defaults to 1 (`step` should not be zero)
  - If the `start` argument is omitted, it defaults to 0

```
range(5)                # 0, 1, 2, 3, 4
range(-1, 4)            # -1, 0, 1, 2, 3
range(0,10,2)           # 0, 2, 4, 6, 8
range(5,0,-1)           # 5, 4, 3, 2, 1
range(10,2)             # ???
```

- `list(range(100))` → [0, 1, 2, …, 99]

Looping Through a List (1)

```
print('Prime numbers')
for p in [2, 3, 5, 7, 11, 13, 17, 19]:
    print(p)
```

```
for name in ['Liam', 'Noah', 'William', 'James']:
     print('Hi,', name)
```

Looping Through a List (2)

```
friends = ['Harry', 'Sally', 'Tom', 'Jerry']

for friend in friends:
    print('Merry Christmas,', friend)

for i in range(len(friends)):
    print('Merry Christmas,', friends[i])
```

Finding the Largest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for n in [24, 12, 4, 19, 31, 27]:
    if n > largest_so_far:
        largest_so_far = n
    print(n, largest_so_far)
print('After', largest_so_far)
```

```
Before -1
24 24
12 24
4 24
19 24
31 31
27 31
After 31
```

Counting in a Loop

```
count = 0
print('Before', count)
for n in [24, 12, 4, 19, 31, 27]:
    count = count + 1
    print(n, count)
print('After', count)
```

```
Before 0
24 1
12 2
4 3
19 4
31 5
27 6
After 6
```

Summing in a Loop

```
sum = 0
print('Before', sum)
for n in [24, 12, 4, 19, 31, 27]:
    sum = sum + n
    print(n, sum)
print('After', sum)
```

```
Before 0
24 24
12 36
4 40
19 59
31 90
27 117
After 117
```

Finding the Average

```
sum = 0
count = 0
print('Before', count, sum)
for n in [24, 12, 4, 19, 31, 27]:
    count = count + 1
    sum = sum + n
    print(n, sum, count)
print('After', count, sum, sum/count)
```

```
Before 0 0
24 24 1
12 36 2
4 39 3
19 58 4
31 89 5
27 116 6
After 6 117 19.5
```

Filtering in a Loop

```
print('Before')
for n in [24, 12, 4, 19, 31, 27]:
    if n > 20:
        print(n)
print('After')
```

```
Before
24
31
27
After
```

Searching an Element

```
index = 0
pos = 0
print('Before', pos, index)
for n in [24, 12, 4, 19, 31, 27]:
    if n == 19:
        index = pos
        break
    pos = pos + 1
    print(n, pos, index)
print('After', pos, index)
```

```
Before 0 0
24 1 0
12 2 0
4 3 0
After 3 3
```

Finding the Smallest Value

```
smallest_so_far = -1
print('Before', smallest_so_far)
for n in [24, 12, 4, 19, 31, 27]:
    if n < smallest_so_far:
        smallest_so_far = n
    print(n, smallest_so_far)
print('After', smallest_so_far)
```

```
Before -1
24 -1
12 -1
4 -1
19 -1
31 -1
27 -1
After -1
```

Finding the Smallest Value (Revised)

```
smallest_so_far = None
print('Before')
for n in [24, 12, 4, 19, 31, 27]:
    if smallest_so_far is None:
        smallest_so_far = n
    elif n < smallest_so_far:
        smallest_so_far = n
    print(n, smallest_so_far)
print('After', smallest_so_far)
```

```
Before
24 24
12 12
4 4
19 4
31 4
27 4
After 4
```

Exceptions

- Errors detected during execution even if a statement or expression is syntactically correct
  - `ZeroDivisionError`
  - `NameError`
  - `TypeError`
  - `ValueError`
  - `IndexError`…

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> int('what')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with
base 10: 'what'
```

Handling Exceptions

- Surround a dangerous section of code with `try` and `except`
- If the code in the `try` works – the `except` is skipped
- If the code in the `try` fails – it jumps to the `except` code block

```
x = int(input('Enter a number: '))
x1 = x + 1
print(x, '+ 1 =', x1)
```

```
while True:
    try:
        x = int(input('Enter a number: '))
        break
    except:    # catch all exceptions
        print('Oops, try again...')
x1 = x + 1
print(x, '+ 1 =', x1)
```

Example

```
s = input('Enter a number: ')
try:
    i = int(s)
except:
    i = None

if i is None:
    print('Not a number')
else:
    print('Nice work')
```