python™

Week 5

`while` loops; logic; random numbers; tuples

while Loops

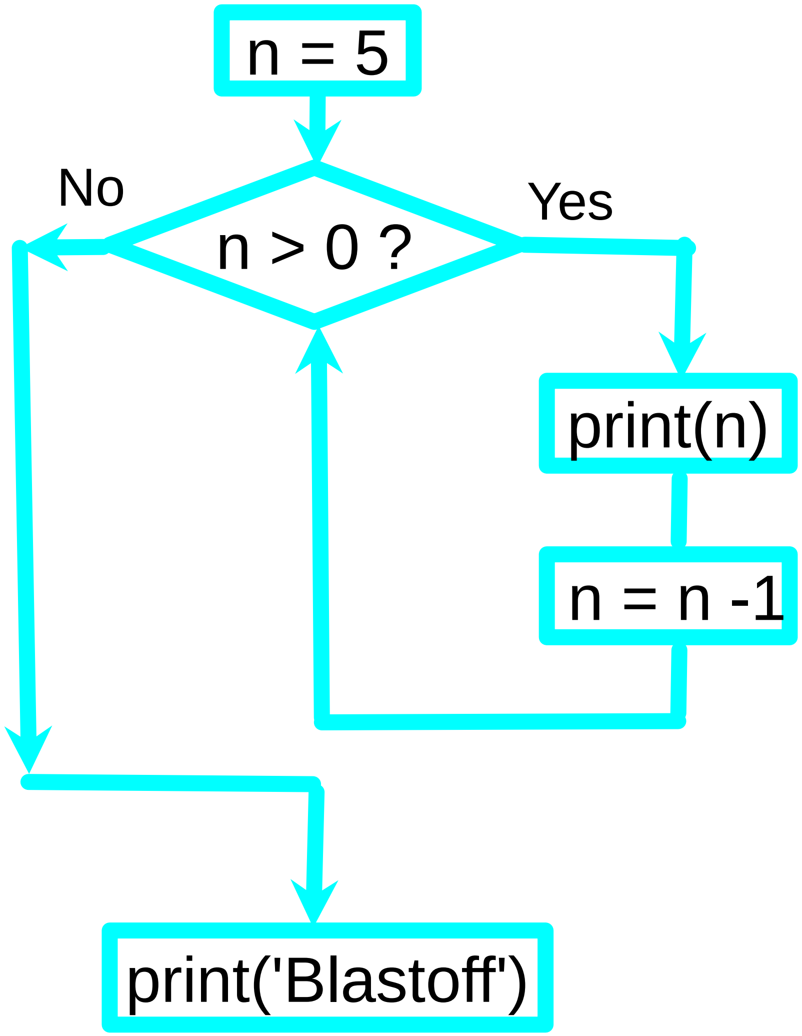
```
while test:
      statements
```

```
>>> n = 91
>>> factor = 2        # find first factor of n

>>> while n % factor != 0:
...     factor += 1
...

>>> factor
7
```

Repeated Steps



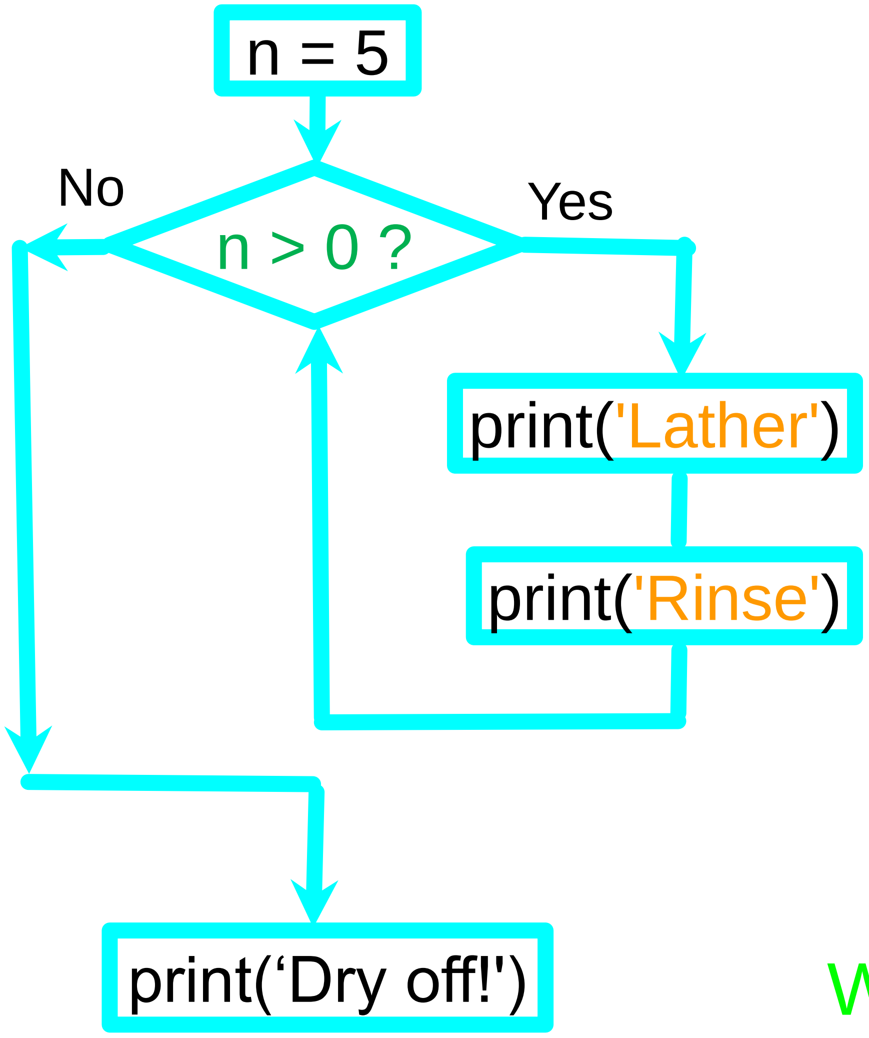
Program:

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

Output:

```
5
4
3
2
1
Blastoff!
0
```

An Loop

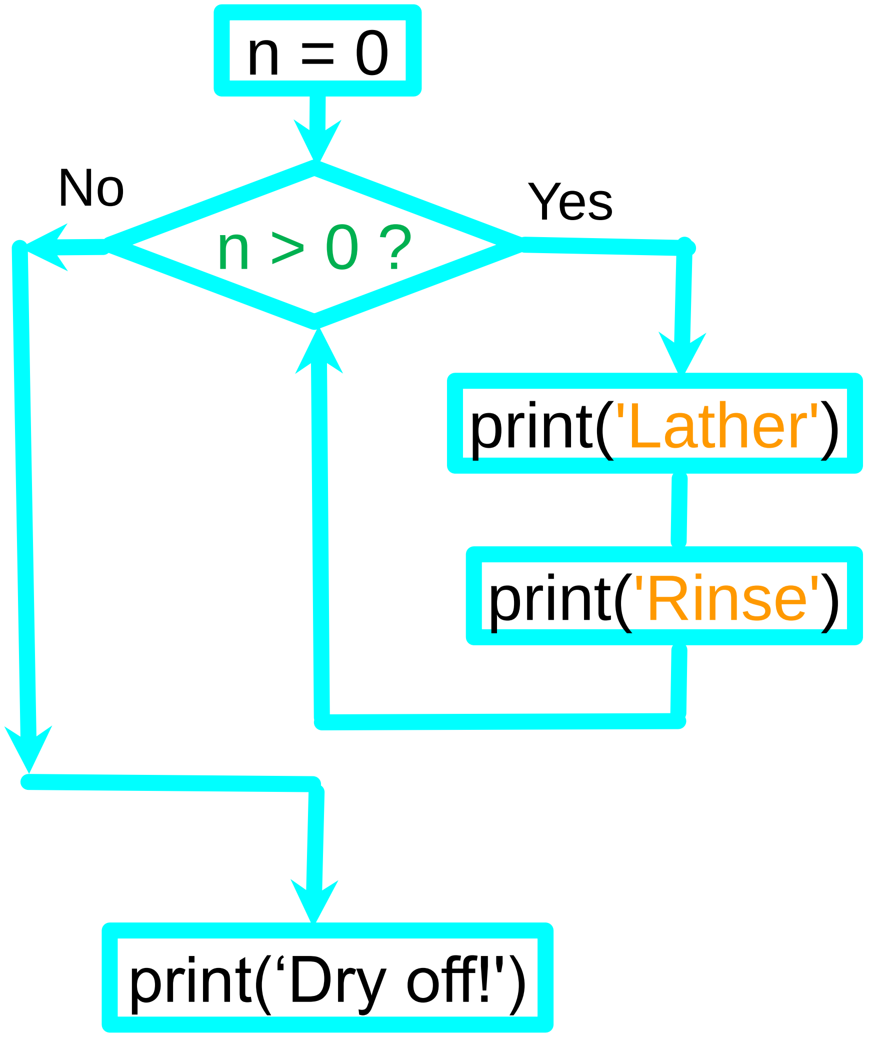


Program:

```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Blastoff!')
print(n)
```

What is wrong with this loop?

Another Loop



Program:

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
print('Blastoff!')
print(n)
```

What is this loop doing?

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!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

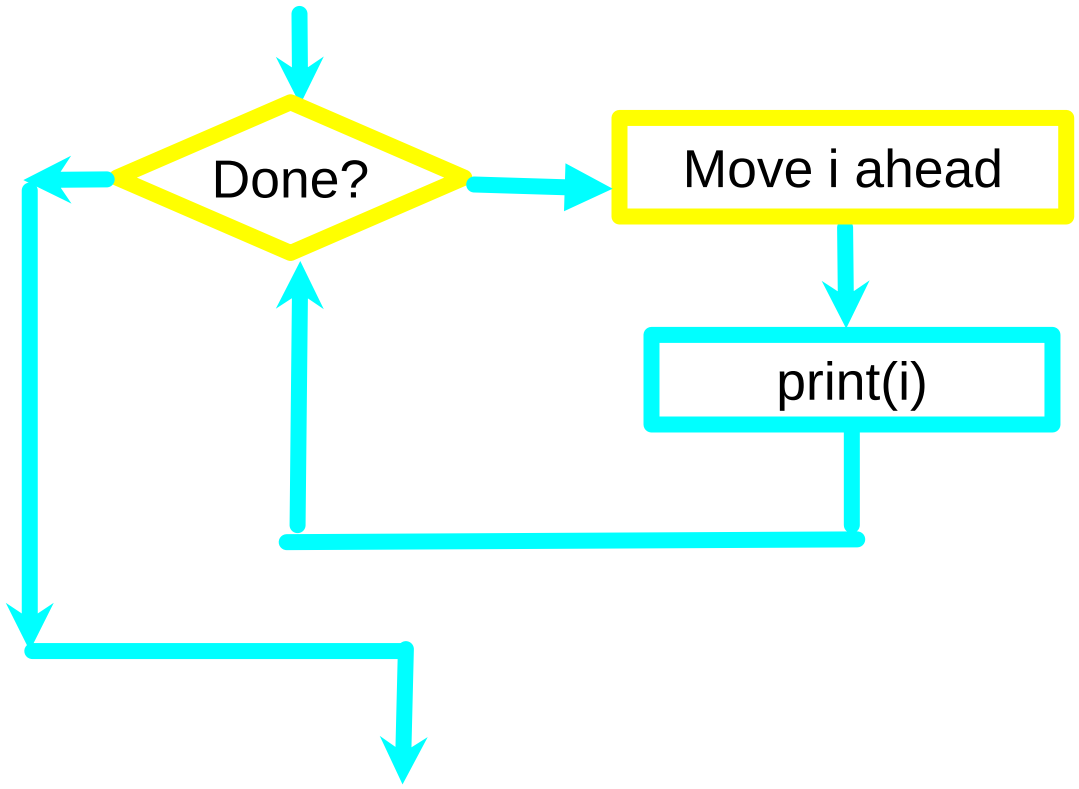
Finishing an Iteration with continue

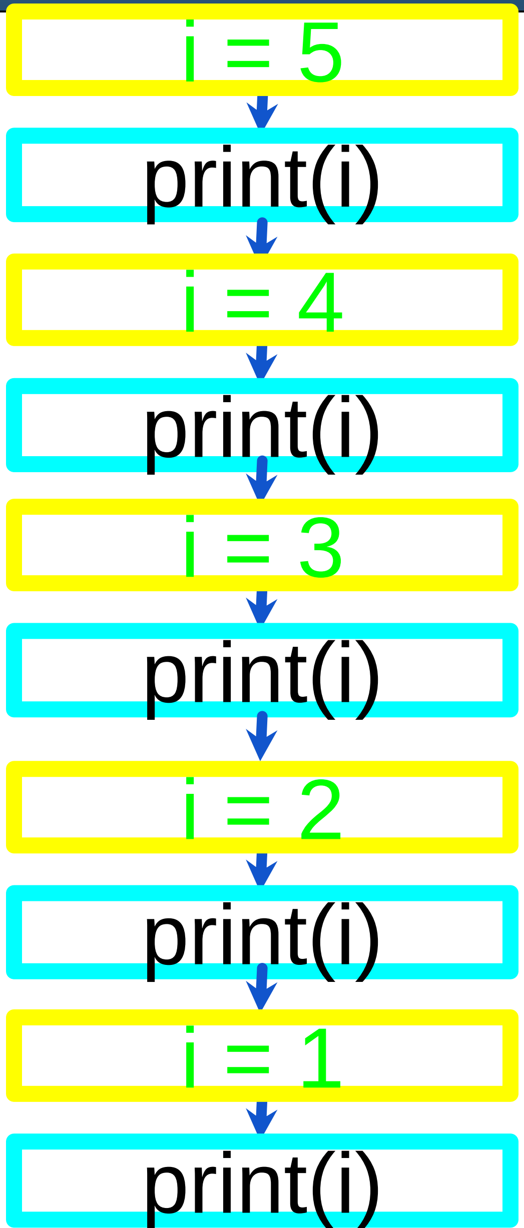
The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

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

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
```

Definite Loops





```
for i in [5, 4, 3, 2, 1]
    print(i)
```

while / else

```
while test:
        statements
else:
        statements
```

- Executes the `else` part if the loop does not enter
- There is also a similar `for` / `else` statement

```
>>> n = 91
>>> while n % 2 == 1:
...     n += 1
... else:
...     print n, "was even; no loop."
...
92 was even; no loop.
```

bool

- Python's logic type, equivalent to `boolean` in Java
  - `True` and `False` start with capital letters

```
>>> 5 < 10
True

>>> b = 5 < 10
>>> b
True

>>> if b:
...     print "The bool value is true"
...
The bool value is true

>>> b = not b
>>> b
False
```

Logical Operators

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Operator	Example	Result
`and`	`2 == 3 and -1 < 5`	`False`
`or`	`2 == 3 or  -1 < 5`	`True`
`not`	`not -1 < 5`	`False`

python™

Random Numbers

`from random import *`

`randint(`**min**`, `**max**`)`

- returns a random integer in range [**min**, **max**] inclusive

`choice(`**sequence**`)`

- returns a randomly chosen value from the given sequence
  - the sequence can be a range, a string, ...

```
>>> from random import *
>>> randint(1, 5)
2
>>> randint(1, 5)
5
>>> choice(range(4, 20, 2))
16
>>> choice("hello")
'e'
```

Exercise

```
$ python countloop.py
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

Exercise

```
$ python countloop.py
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

Exercise

1. Finding the average value of elements in a set
2. Filtering elements in a set that greater than a key value
3. Find the Smallest Value in a set

Exercise

- Write the `Dice` program Python to simulate the rolling of two dice until the sum of the dice is 7:

```
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
You won after 5 tries!
```

Tuple

tuple_name = (**value**, **value**, **...**, **value**)

- A way of "packing" multiple values into one variable

```
>>> x = 3
>>> y = -5
>>> p = (x, y, 42)
>>> p
(3, -5, 42)
```

name, **name**, **...**, **name** = **tuple_name**

- "unpacking" a tuple's contents into multiple variables

```
>>> a, b, c = p
>>> a
3
>>> b
-5
>>> c
42
```

Using Tuples

- Useful for storing multi-dimensional data (e.g. (x, y) points)

```
>>> p = (42, 79)
```

- Useful for returning more than one value

```
>>> from random import *
>>> def roll2():
...     die1 = randint(1, 6)
...     die2 = randint(1, 6)
...     return (die1, die2)
...
>>> d1, d2 = roll2()
>>> d1
6
>>> d2
4
```

Tuple as Parameter

`def` **name**((**name**, **name**, **...**, **name**), **...**):
    statements

– Declares tuple as a parameter by naming each of its pieces

```
>>> def slope((x1, y1), (x2, y2)):
...     return (y2 - y1) / (x2 - x1)
...
>>> p1 = (2, 5)
>>> p2 = (4, 11)
>>> slope(p1, p2)
3
```

Tuple as Return

```
def name(parameters):
    statements
    return (name, name, ..., name)
```

```
>>> from random import *
>>> def roll2():
...     die1 = randint(1, 6)
...     die2 = randint(1, 6)
...     return (die1, die2)
...
>>> d1, d2 = roll2()
>>> d1
6
>>> d2
4
```

Higher Order Functions

- `filter(func, sequence)` returns all values in sequence for which `func(value)` returns `True`

```
>>> def close(p1):
        p2 = (0, 0)
        return dist(p1, p2) < 7

n = ((1, 3), (4, 45), (65, 5))

>>> print (list(filter(close, n)))
[(1, 3)]
```

Exercise

- Write a program that looks for perfect numbers less than or equal to n.
- A perfect number is defined as one that is equal to the sum of its divisors other than itself.
- For example:
  - the divisors of 6 are [1, 2, 3, 6]
  - If you exclude 6, the other divisors add up to 6 (1 + 2 + 3)