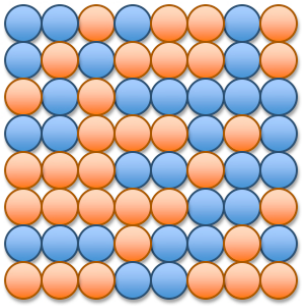

UNIT 2B
An Introduction to Programming (`for` loops)

`for` Loop (simple version)

```
for loop_variable in range(n):
    loop body
```

- The loop variable is a new variable name
- The loop body is one or more instructions that you want to repeat.
- If `n` > 0, the `for` loop repeats the loop body `n` times.
- If `n` <= 0, the entire loop is skipped.
- Remember to indent loop body

`for` Loop Example

```
for i in range(5):
    print("hello world")
```

```
hello world
hello world
hello world
hello world
hello world
```


What Happens to Loop Variable?

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

```
0
1
2
3
4
```

Detour: some printing options

```
for i in range(5):
    print(i, end=" ")
0 1 2 3 4

for i in range(5):
    print(i, end="")
01234

for i in range(10):
    print(i*2, end=" ")
0 2 4 6 8 10 12 14 16 18
```

Blank space after value printed

No space after value printed

Changing the range

```
for i in range(1, 6):
    print(i, end=" ")
1 2 3 4 5

for j in range(1, 6, 2):
    print(j, end=" ")
1 3 5

for k in range(_____________________):
    print(k, end=" ")
5 11 17 23 29 35
```

Increase by 2 each time

Reminder: Assignment Statements

variable = expression

The expression is evaluated and the result is stored in the variable

- overwrites the previous contents of *variable*.

```
a = 5
```

a: 5

Variables change over time

statement	value of x	value of y
`x = 150`	150	?
`y = x * 10`	150	1500
`y = y + 1`	150	1501
`x = x + y`	1651	1501

Accumulating an answer

```
def compute_sum():
    # sums first 5 positive integers
    sum = 0
    for i in range(1, 6):
        sum = sum + i
    return sum

compute_sum()
=> 15
```


Accumulating an answer

```
def compute_sum():
    # sums first 5 positive integers
    sum = 0
    for i in range(1, 6):
        sum = sum + i
    return sum
```

	i	sum
initialize sum	?	0
iteration 1	1	1
iteration 2	2	3
iteration 3	3	6
iteration 4	4	10
iteration 5	5	15

Generalizing sum

```
def compute_sum(n):
  # sums the first n positive integers
  sum = 0
  for i in range(1, n + 1):
     sum = sum + i
  return sum

sum(6)       => 21
sum(100)     => 5050
sum(15110)   => 114163605
```


Danger! Don't change the loop variable!

```
for i in range(5):
    print(i, end=" ")
    i = 10

0 1 2 3 4
```

Even if you modify the loop variable in the loop, it will be reset to its next expected value in the next iteration.

```
for i in range(1,6):
    i = i * 2
    print(i, end=" ")

2 4 6 8 10
```

NEVER modify the loop variable inside a `for` loop.

Accumulation by multiplying as well as by adding

An epidemic:

```
def compute_sick(n):
    # computes total sick after n days
    total_sick = 1
    newly_sick = 1
    for day in range(2, n + 1):
        # each iteration represents one day
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
    return total_sick
```

Each newly infected person infects 2 people the next day.

An epidemic (cont'd)

```
compute_sick(1)  => 1
compute_sick(2)  => 3
compute_sick(3)  => 7
compute_sick(4)  => 15
compute_sick(5)  => 31
compute_sick(6)  => 63
compute_sick(7)  => 127
compute_sick(8)  => 255
compute_sick(9)  => 511
compute_sick(10) => 1023
compute_sick(11) => 2047
compute_sick(12) => 4095
compute_sick(13) => 8191
compute_sick(14) => 16383
compute_sick(15) => 32767
compute_sick(16) => 65535
compute_sick(17) => 131071
compute_sick(18) => 262143
compute_sick(19) => 524287
compute_sick(20) => 1048575
compute_sick(21) => 2097151
```

In just three weeks, over 2 million people are sick!
(This is what Blown To Bits means by *exponential growth*. We will see important computational problems that get exponentially "harder" as the problems gets bigger.)

Countdown

```
import time
def countdown():
    for i in range(1, 11):
        print(11 - i)
        time.sleep(1) # pauses for 1 sec.

countdown()
⇒10
⇒ 9
⇒ 8
…
⇒ 1
```

Why can't we just use `10, 0` here and print `i` instead?

This value gets smaller as `i` gets bigger.

Countdown! (an easier way)

```
import time
def countdown():
    for i in range(10, 0, -1):
        print(i)
        time.sleep(1) # pauses for 1 sec.

countdown()
⇒10
⇒ 9
⇒ 8
…
⇒ 1
```

Now `i` gets smaller at each iteration.