COMP101 — Week 3
Loops & Iteration

for / while, range, break, patterns

UM6P — SASE

October 31, 2025

Legend for Terms in Code

- **Must know**: use today, required moving forward.
- **Need to learn**: useful soon or slightly advanced variants.
- **Optional**: enrichment, stretch, or alternative approaches.

Quick Recap — What you need to know

- Types: `int`, `float`, `bool`, `str`
- Declare / assign variables, naming rules
- Strings: `indexing with []`, `slicing`, `.lower()`, `.split()`
- Arithmetic operation: `+`, `*` , `//`, `%`
- Basic binary

Today: Repetition powers computers

- The power of a computer is **repetition**

Today: Repetition powers computers

- The power of a computer is **repetition**
- We will learn how to **iterate**

Today: Repetition powers computers

- The power of a computer is **repetition**
- We will learn how to **iterate**
- By the end, you should be familiar with: **while**, **for**, **range**, **break**, **in**, **len** (**continue**)

Algorithm

Challenge: Without a calculator, decide if **9973** is prime.

- Think: what would you *repeat* to be sure?

Algorithm

Challenge: Without a calculator, decide if **9973** is prime.

- Think: what would you *repeat* to be sure?
- How many checks can you do in 30s? In 1s?

Algorithm

Challenge: Without a calculator, decide if **9973** is prime.

- Think: what would you *repeat* to be sure?
- How many checks can you do in 30s? In 1s?
- We'll let a **loop** try the boring part.

Motivation #1 — Is a number prime?

Algorithm

Try to divide n by integers $2, 3, \ldots, n-1$ **(or up to $\sqrt{n}$ for speed)**. If any divides, n is not prime.

Motivation #1 — Is a number prime?

```
n = int(input("n? "))
if n < 2:
    print(False)
else:
    is_prime = True
    for i in range(2, n):
        if n % i == 0:
            is_prime = False
            break
    print(is_prime)
```

Uses **for**, **in**, **range**, **break**.

Prime test — while (variant)

```
n = int(input("n? "))
if n < 2:
    print(False)
else:
    i = 2
    is_prime = True
    while i < n:
        if n % i == 0:
            print(False)
            break
    print(True)
```

Uses **while**, **break**.

for loop anatomy

Definition

Use for when you know the sequence or the iteration count in advance (e.g., range, a string ...).

```
# iterate over an iterable
for item in iterable:
    # body: do something with item
    ...
    if need_to_skip_item:
        continue   # jump to next iteration
    if goal_reached:
        break      # exit loop early
```

Definition

Use while when the number of iterations is unknown and driven by a condition (sentinel input, convergence, search).

while loop anatomy

```
# 1) initialize state
state = initial_value

# 2) loop while condition holds
while condition_based_on(state):
    # 3) body: make progress toward stopping
    ...
    if need_to_skip_step:
        continue
    if emergency_stop:
        break
    # 4) update state so the condition will eventually fail
    state = next_state(state)
```

range(start, stop, step) in 45 seconds

Definition

`range(a,b,s)` generates `a, a+s, a+2s, ...` up to `< b`.

```
for i in range(5):          # 0..4
    pass
for i in range(7, 10):      # 7,8,9
    pass
for i in range(10, 2, -2):# 10,8,6,4
    pass
```

Pitfalls: stop is *exclusive*; negative steps need start > stop; off-by-one errors love `range`.

Motivation #2 — Count vowels in a string using in

Algorithm

Scan characters; if `ch.lower() in "aeiou"`, increment a counter.

```
s = input("text? ")
cnt = 0
for ch in s:
    if ch.lower() in "aeiou":
        cnt += 1
print(cnt)
```

Uses **for**, **in**, **counter pattern**, **normalization with** `.lower()`.

Motivation #2 — Count vowels in a string using indices

Algorithm

Loop over indices; if `s[idx].lower() in "aeiou"`, increment a counter.

```
s = input("text? ")
cnt = 0
for idx in range(len(s)):
    if s[idx].lower() in "aeiou":
        cnt += 1
print(cnt)
```

Uses **for**, **in**, **counter pattern**, **normalization with** `.lower()`.

Definition

Comparisons: `<`, `<=`, `>`, `>=`, `==`, `!=` **Logic:** `not`, `and`, `or`

- **and**: both must be True
- **or**: at least one True
- **not**: flips a boolean

A	B	A and B	A or B
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Binary representation (from lowest bit)

Algorithm

Repeat: divide by 2, store remainder, then reverse at the end. Handle $n = 0$.

```
n = int(input("n? "))
if n == 0:
    print("0")
else:
    bits = ""
    x = n
    while x > 0:
        bits += str(x % 2)
        x //= 2
    print(bits[::-1])
```

Uses **while**, //, **%**, **slicing** `s[::-1]`.

Binary representation (from highest bit)

```
n = int(input("n? "))
if n == 0:
    print("0")
else:
    bits = ""
    x = n
    largest_pow = 1
    while largest_pow * 2 < x:
        largest_pow *= 2
    while x > 0:
        if x >= largest_pow:
            bits += "1"
            x -= largest_pow
        else:
            bits += "0"
        largest_pow //= 2
    print(bits)
```

Heron's Method for $\sqrt{a}$ (max_iters)

Definition

Given $a > 0$, iterate $x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$ until convergence.

Heron's Method for $\sqrt{a}$ (max_iters)

```
a = float(input("a? "))
x = 1
max_iters = 100
iters = 0
while iters < max_iters:
    nxt = 0.5 * (x + a / x)
    x = nxt
    iters += 1
print(x)
```

Uses **while**, **max_iters**.

(Stretch) Approximate π via Riemann sum

Theorem / Fact

$\pi \approx 4 \int_0^1 \sqrt{1-x^2}\, dx$. Approximate area with n rectangles.

(Stretch) Approximate π via Riemann sum

```
n = int(input("n rectangles? "))
w = 1.0 / n
area = 0.0
for k in range(n):
    x = (k + 0.5) * w  # midpoint
    y = (1 - x*x) ** 0.5
    area += y * w
print(4 * area)
```

Uses **for**, **range**, **accumulator**.

(Stretch) Approximate π via Monte Carlo

Algorithm

Sample $(x, y) \in [0, 1]^2$. Fraction inside $x^2 + y^2 \leq 1$ estimates quarter-circle area $\Rightarrow \pi \approx 4 \cdot \frac{\#\text{inside}}{N}$.

(Stretch) Approximate π via Monte Carlo

```
import random
N = int(input("samples? "))
inside = 0
for _ in range(N):
    x, y = random.random(), random.random()
    if x*x + y*y <= 1.0:
        inside += 1
print(4.0 * inside / N)
```

Uses **for**, **accumulator**; **random sampling**.

for vs while — when to use which

for loop

- Know count of iterations (or a sequence)
- Natural counter via `range`
- Can exit early with **break**

while loop

- Don't know how long it takes
- Condition-driven (converge, input, search)
- Guard with **max_iters** to avoid infinite loops.