Accumulators

Purpose

- To use the `for` and `range` while modifying a value within the for loop.
- To code math that looks like this: $x_i = f(x_{i-1})$

Generalization

Assignments

In python, the `=` sign does NOT mean *equals*, but rather it means *assign to*!

example:

```
x = x**2
```

$x = x^2$ ✘ $x \leftarrow x^2$ ✓

which means that the current value of `x` will be replaced with the square of itself.

```
x = 3
x = x**2.   # x is now 9
x = x**2    # x is now 81
```

Series

In **math**, it is not unusual to describe a sequence of numbers using subscripts.

$$a_1, a_2, a_3 \ldots a_i \ldots a_n$$

or to equate a given subscripted number as a function of the previous number

given a_1

$$a_i = f(a_{i-1}) \quad \text{for} \quad i > 1$$

where f is a predefined function.

In **python**:

```
# ====================
# calculate a_4
# ====================
a1: float = float(input("enter starting number: "))
a2: float = f(a1)                 # a2 <--- f(a1)
a3: float = f(a2)                 # a3 <--- f(a2)
a4: float = f(a3)                 # a4 <--- f(a3)
print (f"a_{4} is {a4}")
```

BUT if we have a *huge* list of numbers, our code can become very unwieldy, and it definitely won't be flexible.

So, the question becomes,...

- if we don't need to save the intermediate values, can we make a more generalized code? YES!

```
# ====================
# calculate a_n
# ====================
n: int = int(input("Enter the value of n to calculate a_n: "))

a: float = float(input("what is the value of a_0: "))
for i in range(n-1):
    a = f(a)                     # a_i <--- f(a_{i_1})
print (f"a_{n} is {a}")
```

Examples

Sums

Assume that the user of your program has a series of n numbers that they want to total.

The total of all the numbers would be:

$$t_n = a_1 + a_2 + a_3 + \ldots a_i \ldots + a_n$$

$$t_n = \sum_{i=1}^{n} a_i$$

What is the i^{th} total?

$$t_0 = 0, \quad t_i = a_i + t_{i-1} \quad \text{for} \quad i > 0$$

Coding

The question becomes... how do we translate this weird math stuff into python?

First lets just loop over the numbers, and assign t_i as we go.

Because we don't care about the intermediate values, (t_{i-1}) we don't have to store that information at all.

```
n: int = int(input("How many numbers do you want to add? "))

total: float = 0                                    # t_0 = 0
for i in range(n):                                  # loop from i=0 to i=n-1
    number: float = float("Enter your number ")     # get a_i
    total = total + number                          # t_i <--- t_{i-1} + a_i
print (f"Your total is: {total}")
```

Factorials

Refresher:

$$3! = 3 \times 2 \times 1$$

$$4! = 4 \times 3 \times 2 \times 1 \quad = 4 \times (3 \times 2 \times 1) \quad = 4 \times 3!$$

Definitiion:

$$0! = 1$$

$$m! = m(m-1)! \quad \text{for} \quad m > 0$$

Coding

Again, loop over `n`, adjusting the variable `factorial` as required

```
# ===================================
# Calculating factorials
# ===================================
n: int = int(input("Which number do you want the factorial for? "))

factorial:int = 1                                 # n = 1
for i in range(n):                                # loop from i=1 to i=n
    m = i + 1
    factorial = m * factorial                     # m! <-- m (m-1)!
print (f" )
```

Fibonacci sequence

The Fibonacci numbers may be defined by the following relation:

$$F_1 = 1, \qquad F_2 = 1$$

and

$$F_n = F_{n-1} + F_{n-2} \quad \text{for} \quad n > 1$$

i	1	2	3	4	5	6	7	8
	1	1	2	3	5	8	13	21

Coding

```
# ================================
# Calculating Fibonacci Sequence
# ================================
n: int = int(input("Which 'nth' Fibonacci number do you want? "))

a = 1                         # f_1
b = 1                         # f_2

for i in range(n-2):
  f = a + b                   # f_i     <--- f_{i-1} + f_{i-2}
  a = b                       # f_{i-2} <--- f_{i-1}
  b = f                       # f_{i-1} <--- f

print(f"The {n}th element of the fibonacci sequence is {f}")
```