# Introduction to Programming

Spring 2022

# Objectives

- Accumulating Results
- Limitation of Computer Arithmetic

# Accumulating Results

- Computing sum of numbers or factorial

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$$

$$n! = 1 * 2 * 3 * 4 * \ldots * n$$

# Accumulating Results

- How we could we write a program to do this?
- Factorial Problem
  - Input n
  - Compute factorial n
  - Output factorial of n
- How do we calculate factorial of 6

$$6! = 1 * 2 * 3 * 4 * 5 * 6$$

# Accumulating Results

- What’s really going on?
- We’re doing repeated multiplications, and we’re keeping track of the running product.
- This algorithm is known as an accumulator, because we’re building up or accumulating the answer in a variable, known as the accumulator variable.

# Accumulating Results

- The general form of an accumulator algorithm looks like this:

1) `Initialize the accumulator variable`
2) `Loop until final result is reached`
1) `update the value of accumulator variable`

# Accumulating Results

- It looks like we'll need a loop!

```
fact = 1
for factor in [6, 5, 4, 3, 2, 1]:
    fact = fact * factor
```

- Let's trace through it to verify that this works!

# Accumulating Results

- Why did we need to initialize fact to 1? There are a couple reasons…
  - Each time through the loop, the previous value of fact is used to calculate the next value of fact. By doing the initialization, you know fact will have a value the first time through.
  - If you use fact without assigning it a value, what does Python do?

# Accumulating Results

- Great! But what if we want to find the factorial of some other number?
  - We ask the user for n
  - We can use range(n) in our loop.
- There are three form of range
  - range (n)
  - range (start, n)
  - range (start, n, step)
  - Examples

# Accumulating Results

- Complete Factorial Program
- Calculate factorial for:
  - 12, 13, 100
- Try calculating factorial using Python

# The Limits of Int

- What’s going on?
  - While there are an infinite number of integers, there is a finite range of ints that can be represented.
  - This range depends on the number of bits a particular CPU uses to represent an integer value.

# The Limits of Int

- Typical PCs use 32 bits or 64.
- That means there are $2^{32}$ possible values, centered at 0.
- This range then is $-2^{31}$ to $2^{31-1}$. We need to subtract one from the top end to account for 0.
- But our 13! and 100! is much larger than this. How does it work?

# Handling Large Numbers

- Does switching to float data types get us around the limitations of ints?
- If we initialize the accumulator to 1.0, we get

```
The factorial of 30 is
                   2.652528598121911e+32
```

- We no longer get an exact answer!

# Handling Large Numbers

- Very large and very small numbers are expressed in scientific or exponential notation.
- 2.6525285981219111e+32 means 2.6525285981219111 $* 10^{32}$
- Here the decimal needs to be moved right 32 decimal places to get the original number, but there are only 16 digits, so 16 digits of precision have been lost.

# Handling Large Numbers

- Floats are approximations
- Floats allow us to represent a larger range of values, but with fixed precision.
- Python has a solution, expanding ints!
- Python ints are not a fixed size and expand to handle whatever value it holds.

# Handling Large Numbers

- Newer versions of Python automatically convert your ints to expanded form when they grow so large as to overflow.
- We get indefinitely large values (e.g. 100!) at the cost of speed and memory

# Practice

- Show the sequence of numbers that would be generated by each of the following range expressions:
  - range(5)
  - range (3, 10)
  - range (4, 13, 3)
  - range (15, 5, -2)
  - range (5, 3)
  - range (1, 10, -1)