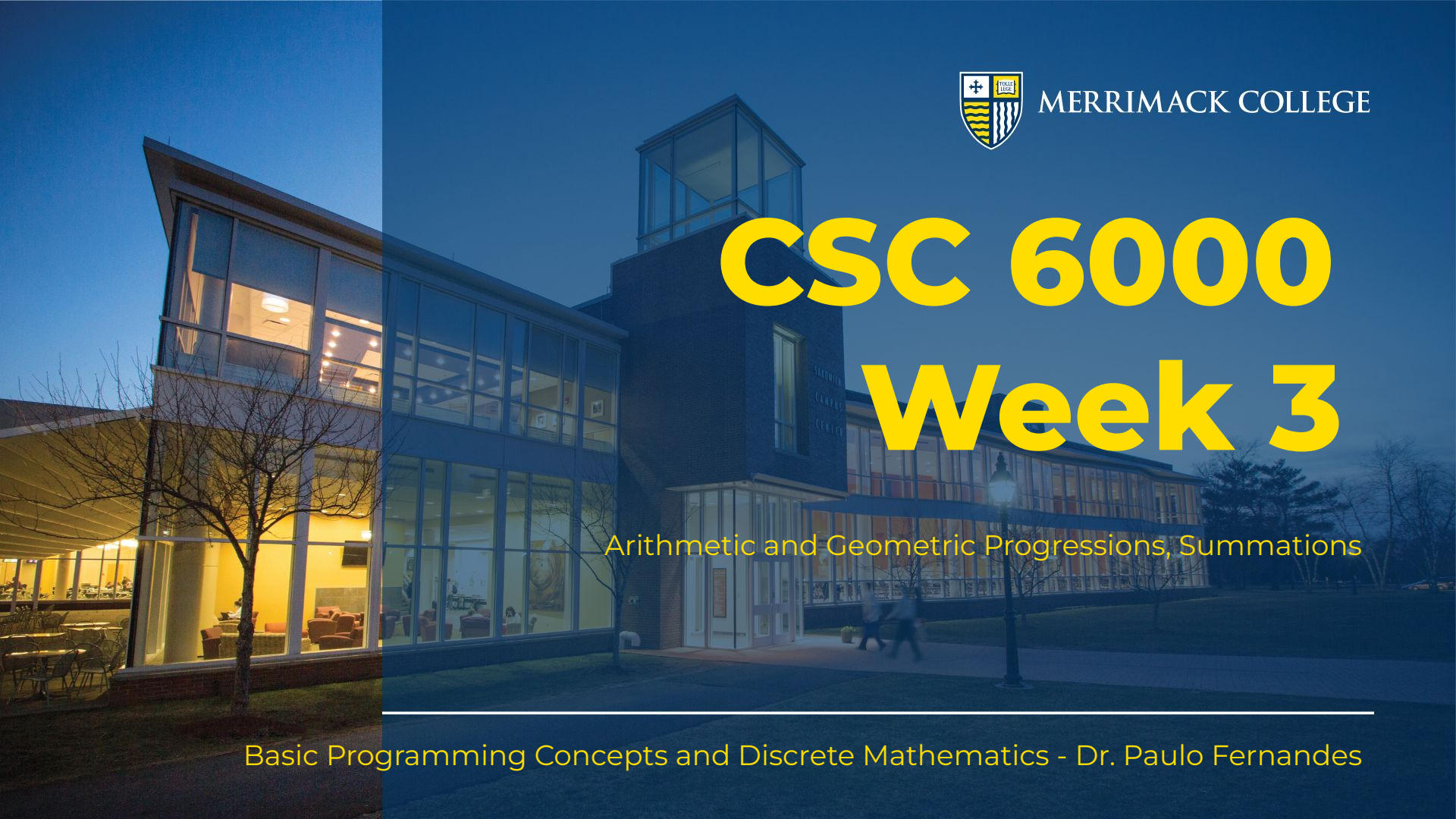


MERRIMACK COLLEGE

CSC 6000 Week 3

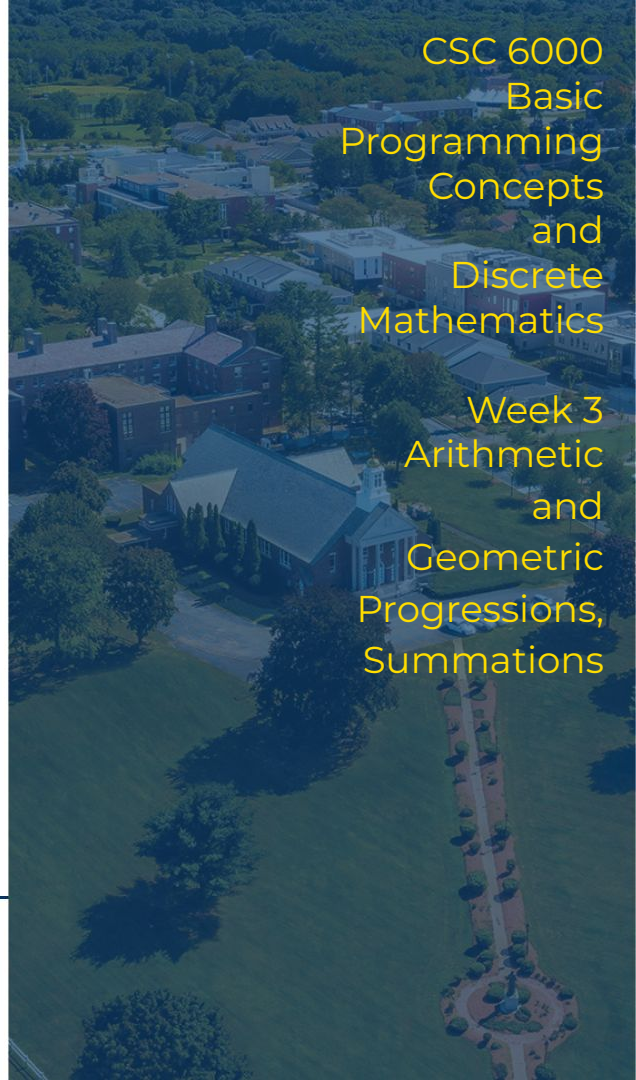
Arithmetic and Geometric Progressions, Summations

Basic Programming Concepts and Discrete Mathematics - Dr. Paulo Fernandes

Presentation Agenda

Week 3

- Arithmetic Progressions (AP)
  a. Sequences
  b. The elements of a AP
  c. The sum of an AP
     i. Summations
- Geometric Progressions (GP)
  a. The elements of a GP
  b. The sum of a GP
  c. Series
- This Week's tasks



Arithmetic Progressions

1, 2, 3, 4 ... 5, 6, 7, 8, 9, and 10

Feist, 2007

Let's talk about sequences! As a sequence is any form of ordered elements, a Python list is a sequence.

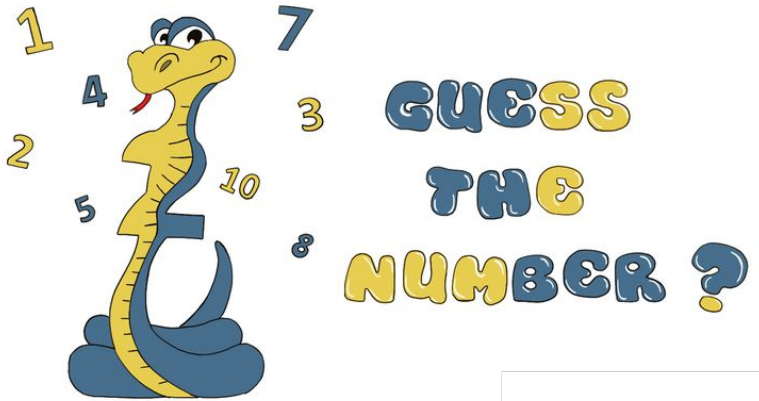
Among the sequences we are interested in are the ones that can be defined recursively, i.e., a bunch of numbers that I can derive from some knowledge on how they are built.

One of the simplest sequences defined by recursion are the arithmetic progressions (AP):

- **Sequences**
- The elements of a AP
- The sum of an AP

Sequences (defined by recursion)

- If I say to you **one**, **two**, **three**
  - What do you say?
- If I say to you **two**, **four**, **six**
  - What do you say?
- If I say to you **fourteen**, **ten**, **six**
  - What do you say?



Those above are simple series, but if I say:

- 3, 6, 12, what do you say?
- 7, 7, 7, what do you say?
- 345, 5865, 99705, what do you say?
- ½, ¼, ⅛ , what do you say?
- 1, 1, 2, 3, 5, 8, what do you say?
- 2, 3, 5, 7, 11, 13, what do you say?

All those (but the last) are sequences defined by recursion, because you know what comes next.

Sequences

```
print(   list(range(1,5))      ,   end="\n\n")
print(   list(range(2,10,2))   ,   end="\n\n")
print(   list(range(14,0,-4))  ,   end="\n\n")
```

```
[1, 2, 3, 4]

[2, 4, 6, 8]

[14, 10, 6, 2]
```

So simple that it just takes a range call

- If I say to you **one**, **two**, **three**
  - What do you say?
- If I say to you **two**, **four**, **six**
  - What do you say?
- If I say to you **fourteen**, **ten**, **six**
  - What do you say?

If a sequence can be defined by taking the last number and adding a number to it, it is an **arithmetic progression (AP)**.

If a sequence can be defined by taking the last number and multiplying it by a number, it is a **geometric progression (GP)**.

a.k.a. Progressions

Wikipedia: Sequence.

Progressions

Try it out:

- Make a Python program that prints 10 elements of an arithmetic progression starting with 100 and each time it has 1 more (100, 101, 102, ...);
- Make a Python program that prints 12 elements of an arithmetic progression starting with 1000 and each time it has 10 less (1000, 990, 980, ...);
- Make a Python program that prints 7 elements of an arithmetic progression starting with 5605 and each time it has more 17 (5605, 5622, 5639, ...);

```
for i in range(100, 110):
    print(i)

for i in range(1000, 880, -10):
    print(i)

for i in range(5605,5724,17):
    print(i)
```

Try it!

I don't know about you, but for me the hard thing is to figure out when to stop!

a.k.a. Sequences

Arithmetic progressions are easy to build, but frequently overlooked.

Arithmetic Progressions

1, 2, 3, 4 ... 5, 6, 7, 8, 9, and 10

Feist, 2007

Let's talk about sequences! As a sequence is any form of ordered elements, a Python list is a sequence.

Among the sequences we are interested in are the ones that can be defined recursively, i.e., a bunch of numbers that I can derive from some knowledge on how they are built.

One of the simplest sequences defined by recursion are the arithmetic progressions (AP):

- Sequences
- **The elements of a AP**
- The sum of an AP

Generating APs in Python

The generation of an AP is simple, you just need the initial term, let's call it a_1, and the common difference, let's call it d.

But to generate a finite AP you need to know when to stop:

```
for i in range(100, 110):
    print(i)

for i in range(1000, 880, -10):
    print(i)

for i in range(5605,5724,17):
    print(i)
```

- Either by defining the **last element** value:
  - $a_1 = 100$ $d = 1$ just 10 elements
  - $a_1 = 1000$ $d = -10$ just 12 elements
  - $a_1 = 5605$ $d = 17$ just 7 elements

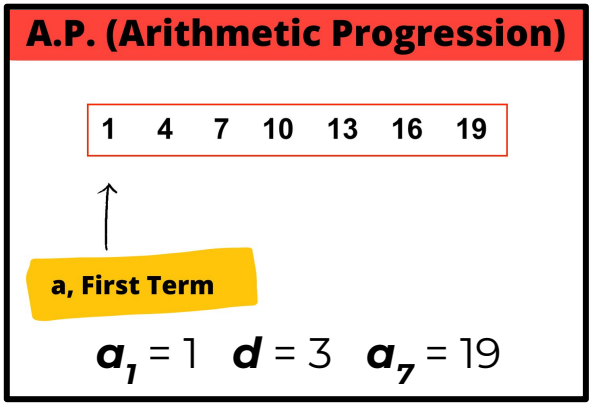
```
[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

[1000, 990, 980, 970, 960, 950, 940, 930, 920, 910, 900, 890]

[5605, 5622, 5639, 5656, 5673, 5690, 5707]
```

- Or by defining the number of elements, but we will see that later.

Note that in Python the first element is **0**, while in Math we usually call the first element 1

The *n*-th element of an AP

- How to find the ***n***-th element of a AP?
  - An AP is defined by:
    - a_1 the initial term, and
    - d the common difference.
- Assuming the initial term a_1 and common difference d:
  - a_1 $\quad a_1 = a_1 + d * 0$
  - $a_2 = a_1 + d$ $\quad a_2 = a_1 + d * 1$
  - $a_3 = a_1 + d + d$ $\quad a_3 = a_1 + d * 2$
  - $a_4 = a_1 + d + d + d$ $\quad a_4 = a_1 + d * 3$



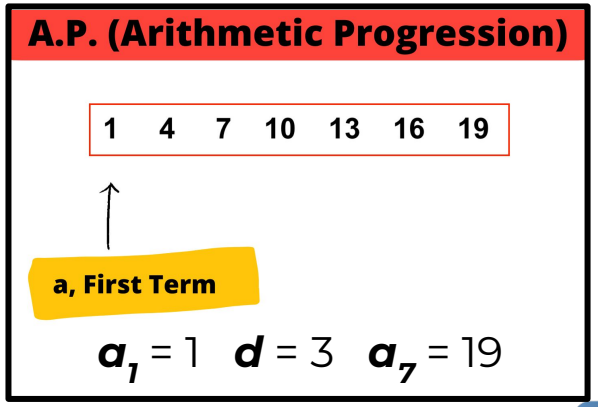
- Thus:
  - $a_n = a_1 + d * (n-1)$

```
def n_th(a1, d, n):
    return a1 + (n-1) * d
```


Wikipedia: Arithmetic Progression.

Generating APs

- How to find the ***n***-th element of a AP?
  - An AP is defined by:
    - a_1 the initial term, and
    - d the common difference.
- The ***n***-th element of such AP is (assuming the first element indexed by 1) is given by:
  - $a_n = a_1 + (d * (n - 1))$
    - for example, giving the AP with $a_1 = 5605$ $d = 17$
      - $a_8 = 5605 + (17 * (8 - 1)) = 5605 + (17 * 7) = 5605 + 119 = 5724$




```
def n_th(a1, d, n):
    return a1 + (n-1) * d

print("The 8th element:", n_th(5605, 17, 8))
```


Wikipedia: Arithmetic Progression.

Generating APs in Python

```
def n_th(a1, d, n):
    return a1 + (n-1) * d

print("The 8th element:", n_th(5605, 17, 8))
```

Knowing how to compute the ***n***-th element of an AP, it is possible to establish the limit of an AP by finding its last element, or for Python purposes, the element just after the last one.

Given an AP defined by the initial term a_1 and the common difference ***d***, an AP with x elements may be generated by a Python range function like this:

- ***range***(a_1 , a_{x+1} , ***d***)
  - for example, for $a_1 = 5605$ $d = 17$ with $x = 7$ elements, we compute $a_8 = 5724 = 5605 + (7 * 17)$ thus:

```
print(  list(  range(5605, 5724, 17)   ) )
```

```
[5605, 5622, 5639, 5656, 5673, 5690, 5707]
```

Text: Python range function.

Arithmetic Progressions

1, 2, 3, 4 ... 5, 6, 7, 8, 9, and 10

Feist, 2007

Let's talk about sequences! As a sequence is any form of ordered elements, a Python list is a sequence.

Among the sequences we are interested in are the ones that can be defined recursively, i.e., a bunch of numbers that I can derive from some knowledge on how they are built.

One of the simplest sequences defined by recursion are the arithmetic progressions (AP):

- Sequences
- The elements of a AP
- **The sum of an AP**

A Brief Tale about Young Gauss

This is an anecdotal tale that may very well be not true. It is said that at the end of the eighteenth century the young **Johann Carl Friedrich Gauss**, the future mathematical genius, was at school and his teacher asked the class to compute the sum of 1 plus 2, plus 3, plus 4, and so on up to 20.

The teacher was expecting the young 10-year-old kids to spend some time doing the sums, but young Gauss almost immediately answered that the sum was 210.

How did the kid compute it so fast?

1777 - 1803 - 1855

An AP with $a_1 = 1$ $d = 1$ with $n = 20$ elements:

```
def n_th(a1, d, n):
    return a1 + (n-1) * d

print(list(range(1,n_th(1, 1, 21),1)))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```


Wikipedia: Carl Friedrich Gauss.

A Brief Tale about Young Gauss

How did the young Gauss compute it so fast?

```
def sum(k):
    acc = 0
    for i in range(1,k+1):
        acc += i
    print("The sum of 1 to", k, "is", acc)
```

1777 - 1828 - 1855

```
sum(20)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
The sum of 1 to 20 is 210
```

$$1+2+3+...+n=?$$

$$\frac{n(n+1)}{2}$$

An AP with $a_1 = 1$ $d = 1$ with $n = 20$ elements:

The sum is (20 * 21) // 2

```
def sum(k):
    print("The sum of 1 to", k, "is", (k*(k+1))//2)
```

MERRIMACK COLLEGE

This is the basics of the sum of terms of an AP.

Sum of terms of any AP

How can we extend this idea to other APs?

- The sum of the first and last elements repeats itself for the sum of:
  - the second and the second-to-last;
  - the third and the third-to-last;
  - and so on...

$a_1 = 1 \quad d = 1$

$$\frac{n(n+1)}{2}$$

1777 - 1840 - 1855

$a_1 = 3 \quad d = 5$
$n = 8$

```
[3, 8, 13, 18, 23, 28, 33, 38]
[38, 33, 28, 23, 18, 13, 8, 3]
```

```
def n_th(a1, d, n):
    return a1 + (n-1) * d

def sum(a1, d, n):
    acc = (a1 + n_th(a1, d, n)) * n / 2
    print("The sum is:", acc)
```

$$\frac{n(a_1 + a_n)}{2}$$

$a_1 = 3 \quad a_8 = 38$
$n = 8$
$sum = 41 * 8 / 2$

It works for Reals too

```
def sum(a1, d, n):
    acc = 0
    term = a1
    for i in range(n):
        acc += term
        term += d
    print("The sum is:", acc)
```


Wikipedia: Arithmetic Progression.

Summation

$$\sum_{\text{variable}=\text{initial}}^{\text{final}} \text{term}$$

Before continuing, let's set a important mathematical notation:

- The **summation** is defined by the Greek letter capital **sigma** with:
  - one subscript which states the variable and its initial value;
  - one superscript that states the final value; and
  - the term expression.

A summation represents the sum of all term expressions with the variable going from the initial to the final value.

$$\sum_{i=1}^{5} i = 1+2+3+4+5 = 15$$

$$\sum_{i=1}^{5} 2i = 2+4+6+8+10 = 30$$

$$\sum_{i=0}^{5} 2^i = 2^0+2^1+2^2+2^3+2^4+2^5 = 2^6-1 = 63$$

It is pretty much like a for command.

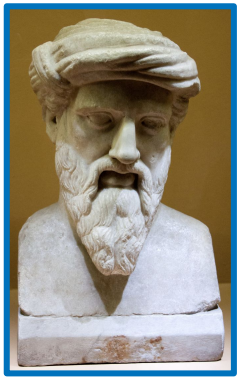
$$\sum_{i=0}^{99} 2i+17$$

```
acc = 0
for i in range(100):
    acc += 2 * i + 17
print("The sum is:", acc)
```

Video: Summation.

Sum of terms of AP, mathematically

Despite Gauss being a genius, the sum of an AP was known, according to several documents, since Pythagoras (five centuries before common era), and by several mathematicians of different cultures (Greek, Indian, China, Arabic, European).



570 bce - 495 bce

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

The Greek letter capital sigma stands for the sum of all terms with a variable ***i*** going from ***1*** to ***n***

$$\sum_{i=0}^{n-1} a_1 + (i \times d) = \frac{n(a_1 + a_n)}{2}$$

The sum of all terms of an AP $\boldsymbol{a_1}$, ***d***, ***n*** with a variable ***i*** going from ***0*** to ***n-1***

Wikipedia: Summation.

Geometric Progressions

The powers of two are a GP!

GPs and powers are quite similar.

Arithmetic progressions (AP) are based on adding, while geometric progressions (GP) are based on multiplying.

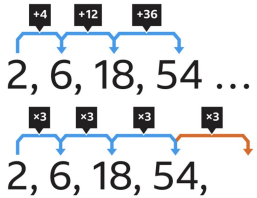
You are able to compute the ***n***-th term of an AP or a GP. There are direct formulas to compute the sum of terms of an AP and of a GP.

However, APs make sense almost *only* when they are finite, while infinite GPs are also useful. Thus we will see:

- **The elements of a GP**
- The sum of an GP
- Series

Progressions

- If I say to you **one**, **two**, **three**
  - What do you say?
- If I say to you **two**, **four**, **six**
  - What do you say?
- If I say to you **fourteen**, **ten**, **six**
  - What do you say?
- If I say to you **one**, **two**, **four**
  - What do you say?
- If I say to you **four**, **six**, **nine**
  - What do you say?
- If I say to you **two**, **one**, **half**
  - What do you say?



If a sequence can be defined by taking the last number and adding a constant to it, then it is an **arithmetic progression (AP)**.

If a sequence can be defined by taking the last number and multiplying it by a constant, it is a **geometric progression (GP)**.

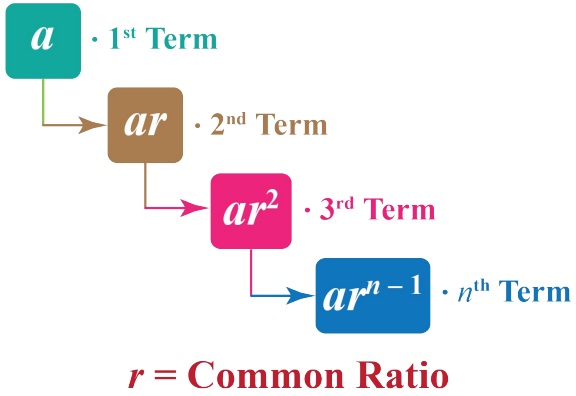
A range is enough…

It is not just a range…

Text: Geometric Sequence.

The *n*-th element of an GP

- Assuming the scale factor **a** (which is also the initial term a_1) and common ratio **r**
  - $a_1 = a$ $\quad a_1 = a * r^0$
  - $a_2 = a * r$ $\quad a_2 = a * r^1$
  - $a_3 = a * r * r$ $\quad a_3 = a * r^2$
  - $a_4 = a * r * r * r$ $\quad a_4 = a * r^3$
- Thus:
  - $a_n = a * r^{(n-1)}$




```
def n_th(a, r, n):
    return a * r ** (n-1)
```


Wikipedia: Geometric Progression.

Geometric Progressions

The powers of two are a GP!

GPs and powers are quite similar.

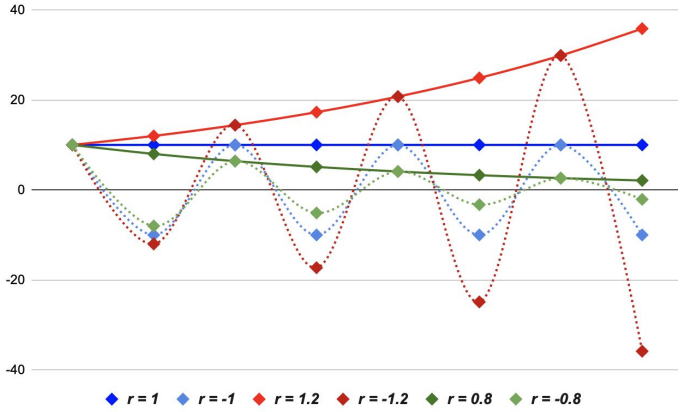
Arithmetic progressions (AP) are based on adding, while geometric progressions (GP) are based on multiplying.

You are able to compute the ***n***-th term of an AP or a GP. There are direct formulas to compute the sum of terms of an AP and of a GP.

However, APs make sense almost *only* when they are finite, while infinite GPs are also useful. Thus we will see:

- The elements of a GP
- **The sum of an GP**
- Series

Sum of terms of a GP



The sum of terms of a GP is computed differently according to the common ratio ***r*** :

- If the absolute value of ***r*** is equal to 1 the GP is actually a **constant** sequence (if ***r = 1***) or a **bipolar** sequence (if ***r = -1***);
- If the absolute value of ***r*** is greater than **1** the GP **increases** indefinitely and we can only compute the sum of a given number of terms (just like an AP);
- If the absolute value of ***r*** is smaller than **1** the GP **decreases** indefinitely and we can either compute the sum of a given number of terms (just like an AP) or the sum of all infinite terms of the GP.

$$\sum_{i=0}^{n-1} ar^i$$

$$a_n = ar^{n-1}$$

```
def n_th(a, r, n):
    return a * r ** (n-1)

def sum(a, r, n):
    acc = 0
    for i in range(n):
        acc += n_th(a, r, i+1)
    print("The sum is:", acc)
```

$$\sum_{i=0}^{n-1} ar^i$$

Text: Sum of a GP.

Sum of terms of a GP

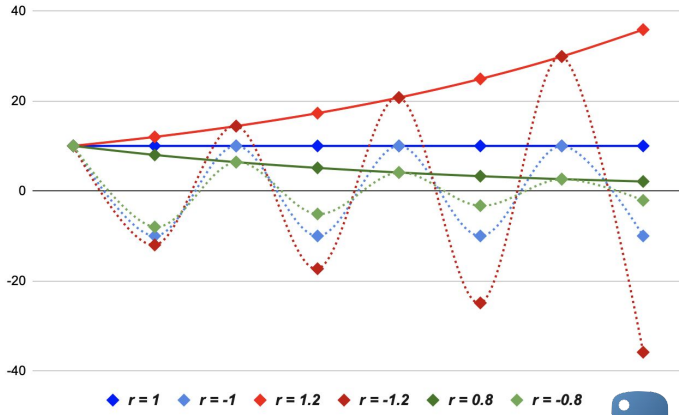
If the absolute value of ***r*** is equal to 1 the GP is actually a **constant** sequence (if ***r = 1***) or a **bipolar** sequence (if ***r = -1***);

- If ***r*** is equal to ***1*** all elements are equal to ***a*** thus the sum of ***n*** elements is given by:

$$\sum_{i=0}^{n-1} ar^i = an$$ [a, a, a, a, a, ...]

- If ***r*** is equal to ***-1*** all elements are alternatively equal to ***a*** and ***-a***, thus either ***a*** or ***0***.

$$\sum_{i=0}^{n-1} ar^i = \begin{cases} a & \text{if } n \text{ is odd} \\ 0 & \text{if } n \text{ is even} \end{cases}$$ [a, -a, a, -a, a, -a, ...]



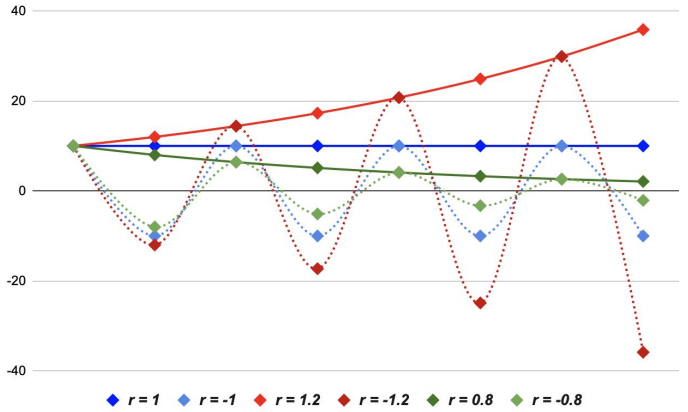

```
def sum(a, r, n):
    if (r == 1):
        print("The sum is:", a * n)
    elif (r == -1) and (n % 2 == 1):
        print("The sum is:", a)
    elif (r == -1) and (n % 2 == 0):
        print("The sum is:", 0)
    else:
        print("Use another formula")
```


Text: Sum of a GP.

Sum of terms of a GP

If the absolute value of ***r*** is greater than ***1*** the GP **increases** indefinitely and we can only compute the sum of a given number of terms (just like an AP) by the formula:

$$\sum_{i=0}^{n-1} ar^i = \frac{a(r^n - 1)}{(r - 1)}$$



Rationale explained in the text referenced below.

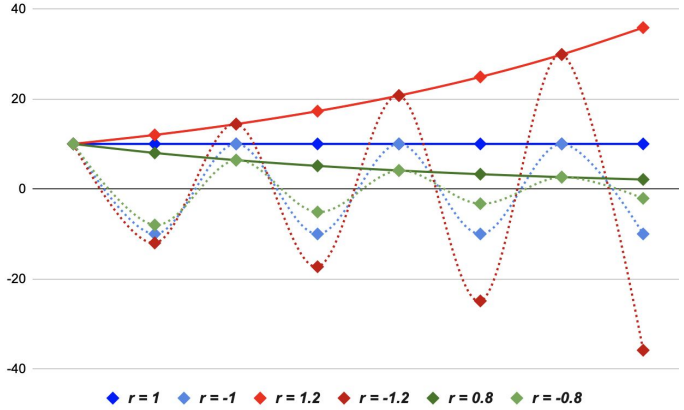
- If a GP with the absolute value of ***r*** greater than 1 has infinite terms, its sum will be infinite as well.

```
def sum(a, r, n):
    print("The sum is:", a * (r ** n - 1) / (r - 1))
```


Text: Sum of a GP.

Sum of terms of a GP

If the absolute value of ***r*** is smaller than **1** the GP **decreases** indefinitely and we can either compute the sum of a given number of terms (just like an AP) or the sum of all infinite terms of the GP.



$$\sum_{i=0}^{n-1} ar^i = \frac{a(r^n - 1)}{(r - 1)}$$

```
def sum(a, r, n):
    print("The sum is:", a * (r ** n - 1) / (r - 1))
```

If the absolute value of ***r*** is **smaller** than ***1*** and the number of terms is infinite (***n*** is closer to infinite), the sum will eventually converge to a finite value, as the terms will diminish it towards 0. Thus:

$$\sum_{i=0}^{\infty} ar^i = \frac{a}{(1 - r)}$$

```
def sum_infinity(a, r):
    if (1 > r > -1):
        print("The sum is:", a / (1 - r))
    else:
        print("The sum is infinite")
```


Text: Sum of a GP.

Geometric Progressions

The powers of two are a GP!

GPs and powers are quite similar.

Arithmetic progressions (AP) are based on adding, while geometric progressions (GP) are based on multiplying.

You are able to compute the ***n***-th term of an AP or a GP. There are direct formulas to compute the sum of terms of an AP and of a GP.

However, APs make sense almost *only* when they are finite, while infinite GPs are also useful. Thus we will see:

- The elements of a GP
- The sum of an GP
- **Series**

Series, Progressions, and Sequences

In general, from a mathematics point of view, you can call a list of numbers a **sequence**.

- If a sequence of numbers can be defined by a recurrence function that involves adding a common difference, it is an **Arithmetic Progression**.
- If a sequence of numbers can be defined by a recurrence function that involves multiplying by a common ratio, it is a **Geometric Progression**.

If your progression is used to express a value that is the sum of its elements, it is called the **series**.

In continuous mathematics, the more relevant series are the infinite series, which are only interesting if the absolute value of the common ratio is smaller than 1.

Video: Sequences and Series.

Series, Discrete Mathematics point of view

GPs that have a common ratio ***r*** smaller than ***1*** are known in mathematics as **series** because they have been used for quite a long time to compute values for several practical applications.

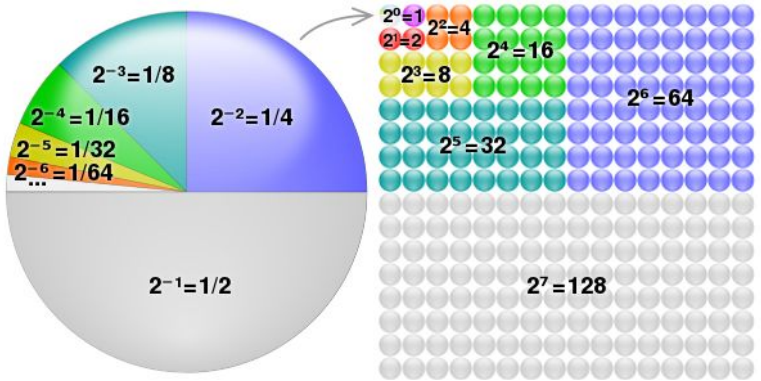
Some series are well known because of their mathematical applications:

- The negative powers of 2

$$\sum_{i=1}^{\infty} 2^{-i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$$

Hence, for ***a*** = ***1*** and ***r*** = ***2***

$$\sum_{i=0}^{n} 2^{i} = 1 + 2 + \cdots + 2^{n} = 2^{n+1} - 1$$



Wikipedia: Series.

This Week's tasks

- Post discussion D#3
- Coding Project P#3
- Quiz Q#3

Tasks

- Post in the discussion how strange progressions were to you, and which of the topics seen today was more surprising to you.
- Coding Project #3, computing sum of GP.
- Quiz #3 about this week topics.

Post Discussion - Week 3 - D#3

Post in the discussion how strange the progressions were to you, and which of the topics seen in this third week was more surprising to you.

- You might be aware of arithmetic progressions, but does it means that you can figure out the sum of APs?
- Which among the topics shown today was more surprising to you?
  - Use personal opinions, but justify your opinion with technical reasons.

Your task:

- Post your answer in the message board by this Monday;
- Reply to posts from your colleagues in the message board by next Thursday.

This task counts towards the Discussions grade.

Third Coding Project - Week 3 - P#3

- Write a Python program that:
  - Asks the user a scale factor ***a*** and a common ratio ***r***;
  - Informs the user if the GP converges to a sum regardless of its number of elements:
    - If the GP converges to a sum regardless of the number of elements, your program should compute the sum of infinite elements;
    - Otherwise, your program should ask the number of elements ***n***, and compute the sum of all elements;
  - With that information, your program should print out the first 3 elements of the GP and the computed sum (example in the next slide).

Your task:

- Go to Canvas, and submit your Python file (.py) within the deadline:
  - The deadline for this assignment is next Thursday.

This assignment counts towards the Projects grade.

Third Coding Project - Week 3 - P#3

- For example, if the user enters: ***a = 10*** and ***r = 0.5***, your program should output:
  - *This GP converges with infinite elements to 20*
  - *the first elements are 10, 5, 2.5*
- For example, if the user enters: ***a = 10*** and ***r = -0.5***, your program should output:
  - *This GP converges with infinite elements to 6.6666667*
  - *the first elements are 10, -5, 2.5*
- For example, if the user enters: ***a = 1*** and ***r = 2***, your program should output:
  - *This GP does not converge to a finite number with infinite elements*;
  - Ask for ***n***, and if the user enters ***n = 10***, the final output should be:
  - *This GP sum with 10 elements is equal to 1023*
  - *the first elements are 1, 2, 4*

Your task:

- Go to Canvas, and submit your Python file (.py) within the deadline:
  - The deadline for this assignment is next Thursday.

This assignment counts towards the Projects grade.

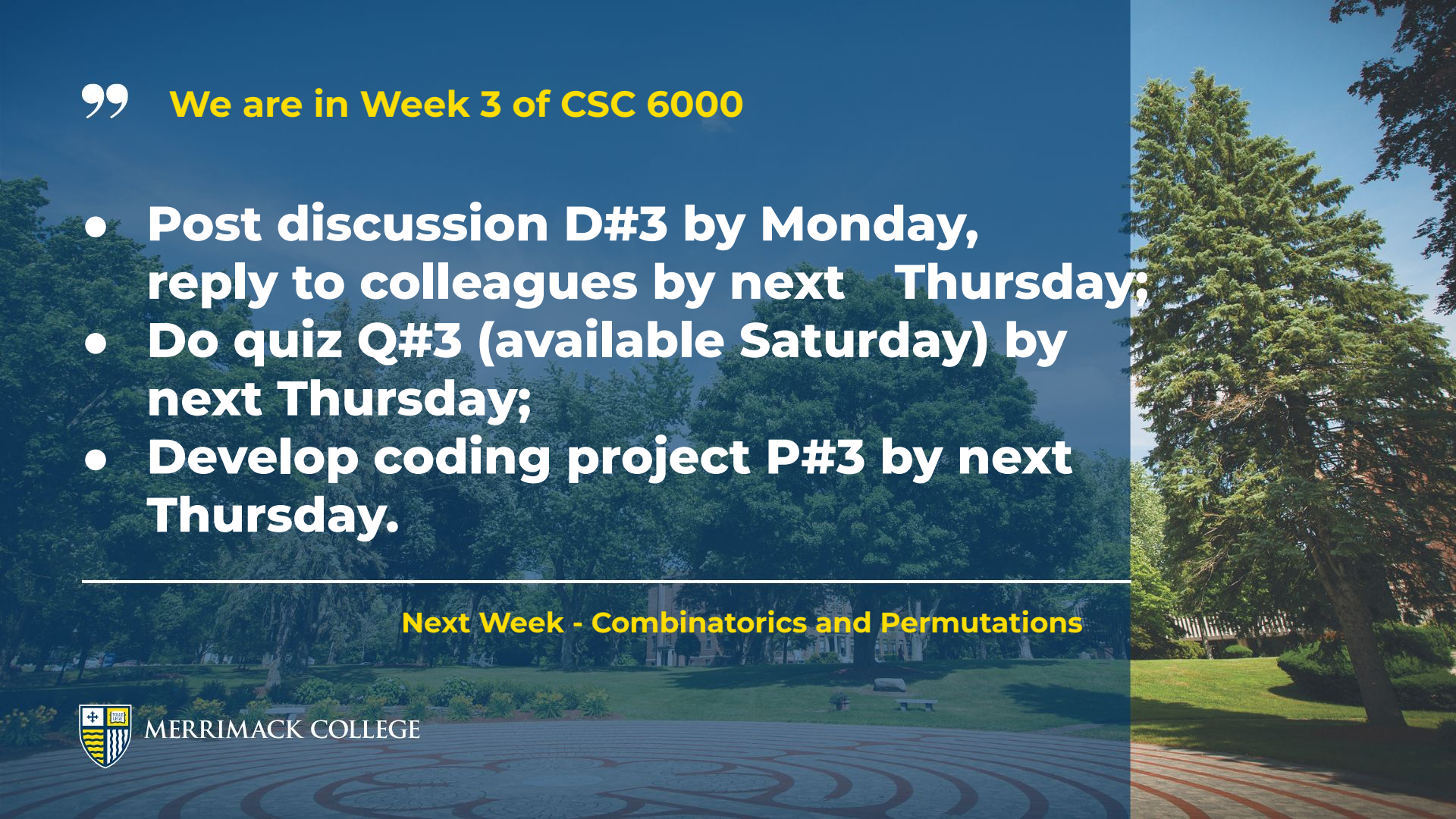
Third Quiz - Week 3 - Q#3

- The third quiz in this course covers the topics of Week 3.
- The quiz will be available this Saturday, and it is composed of 10 questions.
- The quiz should be taken on Canvas (Module 3), and it is not timed.
  - You can take as long as you want to answer it.
- The quiz is open book, open notes, and you can even use any language interpreter to answer it.
- Yet, the quiz is evaluated and you are allowed to submit it only once.

Your task:

- Go to Canvas, answer the quiz and submit it within the deadline:
  - The deadline for the quiz is next Thursday.

This quiz counts towards the Quizzes grade.



We are in Week 3 of CSC 6000

- Post discussion D#3 by Monday, reply to colleagues by next Thursday;
- Do quiz Q#3 (available Saturday) by next Thursday;
- Develop coding project P#3 by next Thursday.

Next Week - Combinatorics and Permutations

MERRIMACK COLLEGE