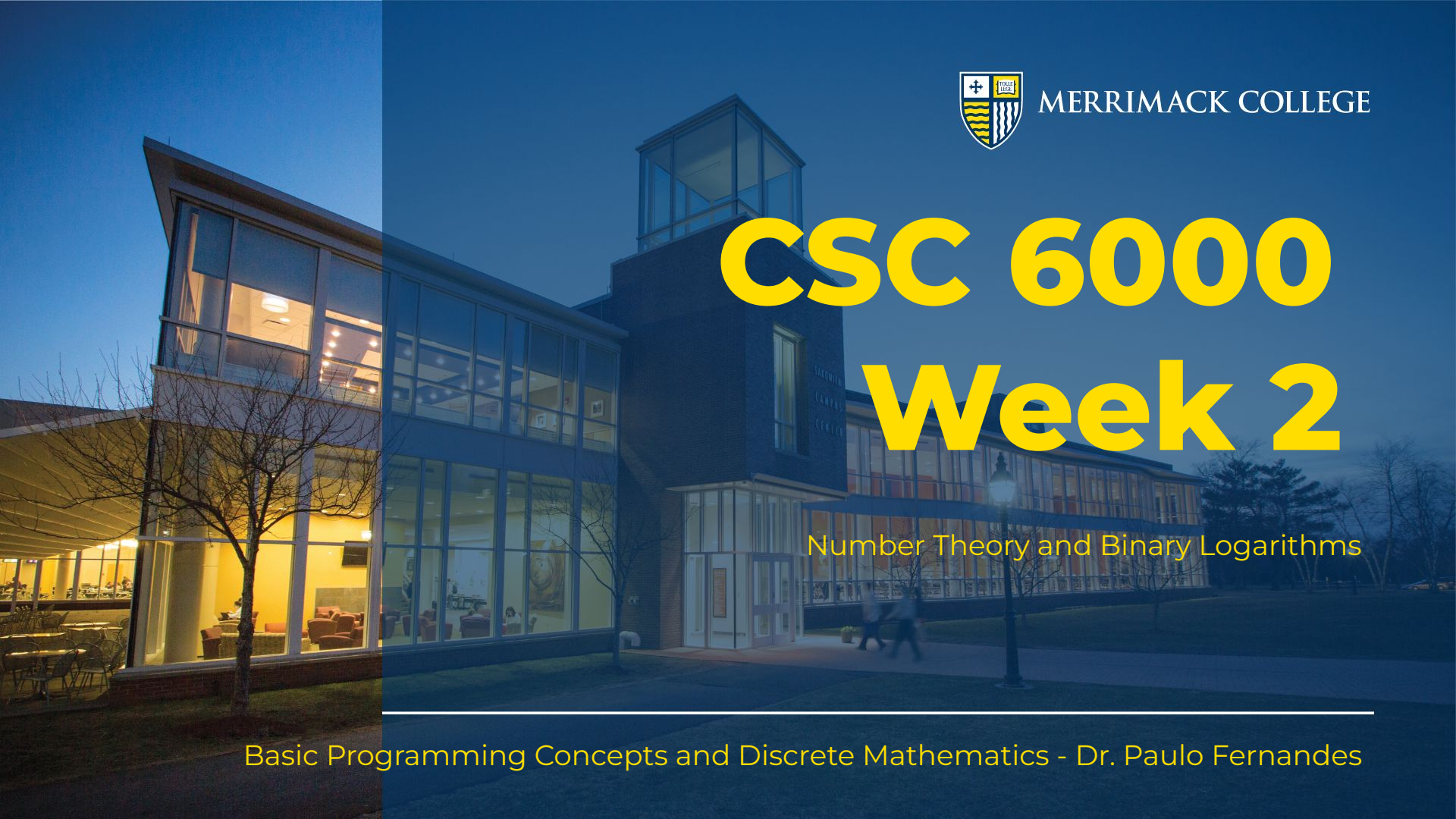


CSC 6000 Week 2

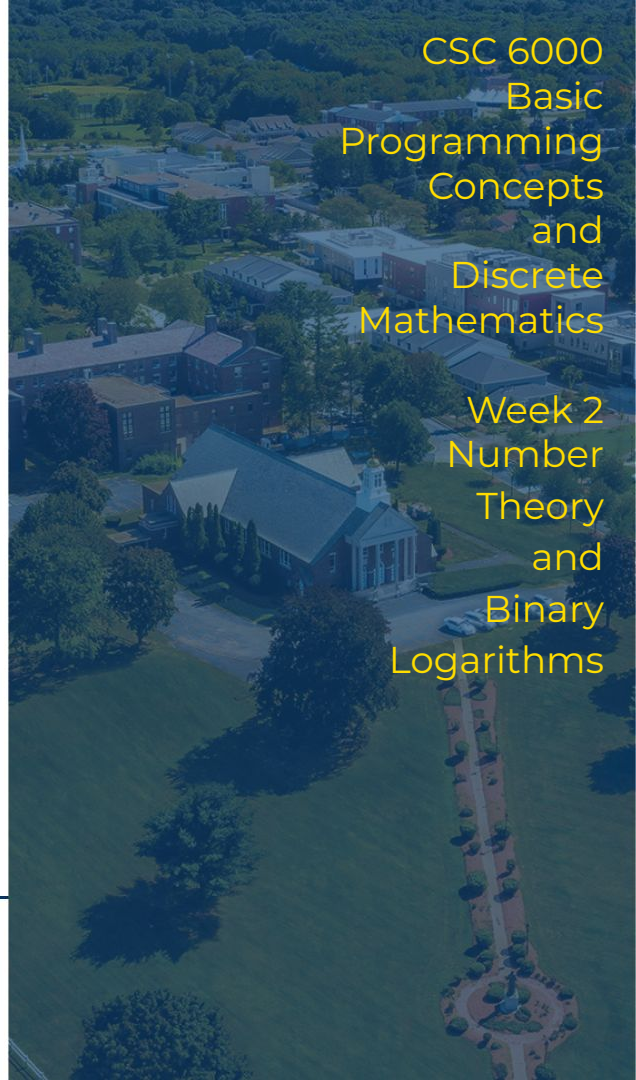
Number Theory and Binary Logarithms

Basic Programming Concepts and Discrete Mathematics - Dr. Paulo Fernandes

Presentation Agenda

Week 2

- Number Theory
  a. Naturals, Integers, and Reals
  b. Primality
     i. Prime and Composite Numbers
     ii. Python Prime Detector
- Number systems
  a. Decimal and Binary
     i. Binary logarithms
  b. Octal and Hexadecimal
- This Week's tasks



Number Theory

"A, B, C, it's easy as 1, 2, 3"

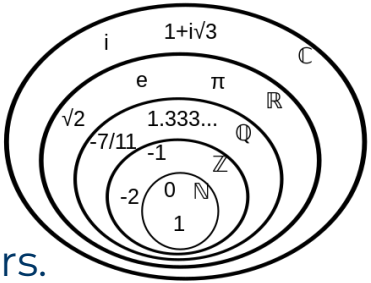
Jackson 5, 1970

Counting is one of the most basic things humans are capable of doing. Natural numbers have intuitively appeared in all human cultures. Most of us never really stop to think about what numbers are, but a lot of mathematicians have!

We are going to pass by the basic concepts of number sets, then we will focus on the concept of Prime numbers and what it entails.

- **Naturals, Integers, and Reals**
- Primality
  - finding primes and multiples

Numbers



Natural numbers were created by God, everything else is the work of men.

Leopold Kronecker
XIX century German Mathematician

ℕ - represents the Natural numbers.
It generally includes 0.
The Romans didn't have it, and they managed to conquer a large empire.

It is preferable to be explicit if you want to consider the zero or not:

- $\mathbb{N}^* = \{ 1, 2, 3, 4, \dots \}$
- $\mathbb{N}_0 = \{ 0, 1, 2, 3, \dots \}$

ℤ - the Integer (whole) numbers

ℚ - the Rational numbers

ℝ - the Real numbers

ℂ - the Complex numbers

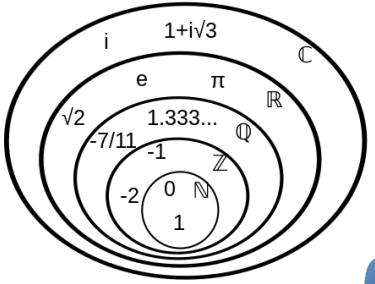
Wikipedia: Numbers.

Numbers

ℕ - very important in discrete mathematics

ℤ - almost all programming languages deal with Integers

ℝ - almost all programming languages deal with floating points, representing Reals



In Python:

- Integers: ***int***
  - ***int(5)***
  - ***int(5.4)***
  - ***int(5.8)***
  - ***int(33 / 4)***
- Reals: ***float***
  - ***float(5)***
  - ***float(33 / 4)***
  - ***float(33 // 4)***

In Python Integers are limitless!

(try ***2 ** 10000***)

In Python, Reals are limited by the precision and size!

(try ***1 / 3***, then try ***(10000 / 3)***, then try ***(1 / 3) * 10000***)

Special cases

inf - ***1.5 ** 10000***
float("inf")
nan - ***1.5 / 0***
float("nan")

Text: Floats in Python.

Number Theory

"A, B, C, it's easy as 1, 2, 3"

Jackson 5, 1970

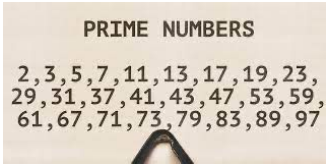
Counting is one of the most basic things humans are capable of doing. Natural numbers have intuitively appeared in all human cultures. Most of us never really stop to think about what numbers are, but a lot of mathematicians have!

We are going to pass by the basic concepts of number sets, then we will focus on the concept of Prime numbers and what it entails.

- Naturals, Integers, and Reals
- **Primality**
  - finding primes and multiples

Prime and Composite Numbers

- We are talking about Natural numbers
  - There is zero (0) and there is one (1)
  - There are **prime** numbers, numbers that exist by themselves. A number is prime if:
    - It is not the product of other numbers;
    - It is only divisible by 1 and itself;
  - There are the composite numbers, numbers that are a product of two or more prime numbers (with repetition):
    - they are:
      - 4 = 2 * 2
      - 6 = 2 * 3
      - 8 = 2 * 2 * 2
      - 9 = 3 * 3
      - 10 = 2 * 5
      - 12 = 2 * 2 * 3
      - 14 = 2 * 7
      - 16 = 2 * 2 * 2 * 2
      - 18 = 2 * 3 * 3
      - 20 = 2 * 2 * 5
      - 21 = 3 * 7
      - ...

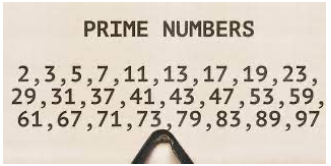


Wikipedia: Prime Number.

Python Prime Detector

How can you make a Python program that discovers if a number is prime?

- You need decisions
  - Python command ***if elif else***
- You might need arrays
  - Python ***lists***
- You need iteration
  - Python **for** or **while** loops
- You need some planning
  - How to do it?
  - How to make it scalable?
  - How to not waste time?



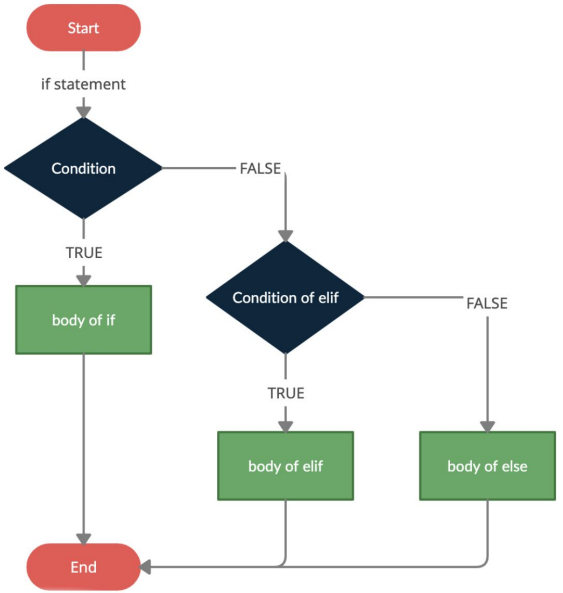

```
num = int(input("Enter an Natural number: "))
if num > 1:
    for i in range(2, int(num/2)+1):
        if (num % i) == 0:
            print(num, "is not a prime number")
            break
    else:
        print(num, "is a prime number")
else:
    print(num, "is not a prime number")
```

(Intentionally too small!)

In programming there is always another right way to do it … and many more wrong ways too.

Python command *if elif else*

- Three usual syntaxes for conditionals:
  - ***if*** *<condition 1>**:***
    <commands if condition 1 is True>
  - ***if*** *<condition 1>**:***
    <commands if condition 1 is True>
    else:
    <commands if condition 1 is False>
  - ***if*** *<condition 1>**:***
    <commands if condition 1 is True>
    elif *<condition 2>**:***
    <commands if condition 2 is True>
    else:
    <commands if all conditions are False>



Text: Conditions in Python.

Python command *if elif else*

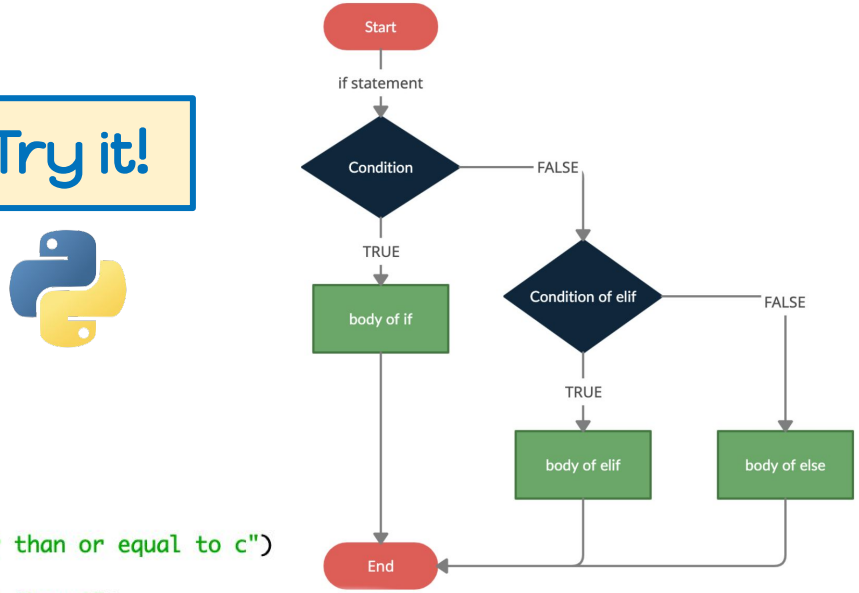
```
a, b, c = 56, 34, 22


if (a > b):
    print("a is greater than b")


if (a < b):
    print("a is smaller than b")
else:
    print("a is greater than or equal to b")


if (a < b + c):
    print("a is smaller than b and c added")
elif (b >= c):
    print("a is greater than b and c added, and b is greater than or equal to c")
else:
    print("a is greater than b and c added, and b is smaller than c")
```

Try it!



There are one-liner versions, but usually this is bad style

Shorthand if - Shorthand if else

Python lists

- A list is an ordered set of variables indexed from 0
  - Python lists of variables of the same kind are arrays:
    - ***a = [0, 11, 22, 33, 44, 55, 66]***
    - ***s = ["first", "second", "third"]***
  - Python lists can also contain different kinds of variables:
    - ***b = ["John", 40, "Paul", 42, "George", 43, "Ringo", 40]***
  - You can access by index just like for strings:
    - ***a[1]*** is the number ***11***
    - ***a[-1]*** is the number ***66***
    - ***s[:2]*** is the list ***["first", "second"]***
    - ***s[1:]*** and ***s[-2:]*** is the list ***["second", "third"]***
    - ***b[0:-1:2]*** is the list ***["John", "Paul", "George", "Ringo"]***
  - Concatenation and product operations are available too.

Text: Python Lists.

Python lists

Try it!

- List have some methods of their own that simplify coding, even though they may be slow sometimes...
  - ***append(<element>)*** it appends the element at the end of the list;
  - ***insert(x, <element>)*** it inserts the element at the index **x**;
  - ***pop(x)*** it removes the element at the index **x**;
  - ***remove(<element>)*** it removes the first occurrence of the element from the list;
  - ***reverse()*** it reverses the order of the elements inside the list;
  - ***sort()*** it sorts the elements in ascending order;
  - ***len(<list>)*** counts the number of elements of a list.

```
a = [0, 11, 22, 33, 44, 55, 66]
s = ["first", "second", "third"]
b = ["John", 40, "Paul", 42, \
     "George", 43, "Ringo", 40]
c = [77, 88, 99]

d = a[1:] + c + a[0:1]
print(d)
print("a:", len(a), "c:", len(c), "d:", len(d))
s.append("fourth")
print(s)
b.insert(2, "Yoko")
b.insert(3, 33)
print(b)
b.pop(0)
b.pop(0)
print(b)
b.remove("George")
b.remove(43)
print(b)
s.reverse()
print(s)
s.sort()
print(s)
print("s:", len(s))
```

Text: Python Lists Methods.

Python *for* Loops

Definite Loop

- ***for*** *<variable>* ***in*** *<list>:*
  <commands>

```
for i in [0,1,2,3,4]:
    print(i)
```

- The *<variable>* will be of the same type as the elements of the *<list>*
- The list can be expressed as:
  - An explicit list
  - A list variable
  - A function returning a list
    - Usually: ***range****(…)*

```
seq = [0,1,2,3,4]
for i in seq:
    print(i)
```

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

- ***range****(<stop>)*
  - ***range(5)*** is equivalent to [0,1,2,3,4]
  - ***range(11)*** is equivalent to [0,1,2,3,4,5,6,7,8,9,10]
- ***range****(<start>,<stop>)*
  - ***range(1,11)*** is equivalent to [1,2,3,4,5,6,7,8,9,10]
  - ***range(5,15)*** is equivalent to [5,6,7,8,9,10,11,12,13,14]
  - ***range(-2,4)*** is equivalent to [-2,-1,0,1,2,3]
- ***range****(<start>,<stop>,<step>)*
  - ***range(1,10,2)*** is equivalent to [1,3,5,7,9]
  - ***range(1,11,2)*** is equivalent to [1,3,5,7,9]
  - ***range(9,-1,-1)*** is equivalent to [9,8,7,6,5,4,3,2,1,0]
  - ***range(15,3,-3)*** is equivalent to [15,12,9,6]

Text: Python range function.

Python *for* Loops

Definite Loop

- ***for*** *<variable>* ***in*** *<list>:*
  <commands>
- Loops are meant to pass by several instances of a list;
  - even a string is a list of characters;
- The use of the built-in function ***range*** can be used to generate values within a list, or to generate indexes to access values of a list;
- Loops can be nested.

Try it!

```
for v in [1, 9, 2, 6, 0, 5, 3, 8, 4, 7]:
    print(v)

a = [5, 3, 8, 4, 7, 1, 9, 2, 6, 0]
for v in a:
    print(v)

for c in "definite_loop":
    print(c)

a = [3, 2, 5, 6, 9, 4, 8, 0, 7, 1]
for i in range(10):
    print(a[i])

for v in range(10):
    print(v)

for v in range(5, 15):
    print(v)

for v in range(0, 20, 2):
    print(v)

a = [3, 2, 5, 6, 9, 4, 8, 0, 7, 1]
for i in range(10):
    print(a[i])

for v in range(9, -1, -1):
    print(a[i])

for v in range(9, -1, -1):
    print(i)

for v in range(5):
    for w in range(4):
        print(v,w)
```


Text: Python For Loops.

Python *while* Loops

Indefinite Loop

- ***while*** *<condition>:*
  <commands>
- The *<commands>* are executed if the *<condition>* is *True* and they keep being repeated until the *<condition>* stops being *True*;
- Everything that can be done with a ***for*** loop can also be done by a ***while*** loop (and vice-versa, because of the ***break*** command).

The ***break*** command

- Used to stop a loop;
- Almost always used together with an ***if*** controlled condition.

```
v = 0
while (v<10):
    print(v)
    v += 1
```

```
for v in range(10):
    print(v)
```

```
a = [3, 8, 2, 0, 7, 5]
for i in range(len(a)):
    if (a[i] == 0):
        break
    else:
        print(a[i])
```

```
a = [3, 8, 2, 0, 7, 5]
i = 0
while (i < len(a)) and (a[i] != 0):
    print(a[i])
```


Text: Python While Loops.

Number Theory

"A, B, C, it's easy as 1, 2, 3"

Jackson 5, 1970

Counting is one of the most basic things humans are capable of doing. Natural numbers have intuitively appeared in all human cultures. Most of us never really stop to think about what numbers are, but a lot of mathematicians have!

We are going to pass by the basic concepts of number sets, then we will focus on the concept of Prime numbers and what it entails.

- **Naturals, Integers, and Reals**
- Primality
  - finding primes and multiples

Is it prime?

Try it for 23, 59, 121, 189871, 191161, 9876542103, and 31381059607.

- Given a user-input number ***n***, how can we compute if it is prime?
  - get the user's input
  - for all numbers smaller than ***n*** and greater than 1, is ***n*** a multiple of this number?
    - ***n*** is a multiple of ***m*** if, and only if, ***n % m*** is equal to zero;
  - if there are no multiples, it is prime!

```
n = int(input("Enter a number: "))
noMultiples = True
for i in range(n-1, 1, -1):
    if (n % i) == 0:
        noMultiples = False
        break
if (noMultiples):
    print(n, "is a prime")
else:
    print(n, "is a composite")
```


Can you do better than that?

Is it prime?

Try it for 23, 59, 121, 189871, 191161, 9876542103, and 31381059607.

- Given a user-input number ***n***, how can we compute if it is prime?
  - for all numbers smaller than ***n*** and greater than 1, is ***n*** a multiple of this number?
    - but ***n*** cannot be multiple of something greater than its half.

```
n = int(input("Enter a number: "))
noMultiples = True
for i in range(n//2, 1, -1):
    if (n % i) == 0:
        noMultiples = False
        break
if (noMultiples):
    print(n, "is a prime")
else:
    print(n, "is a composite")
```


There is always room for improvement!

What is the LCM?

- Given user-input numbers ***n*** and ***m***, how can we compute its **least common multiple** (**LCM**)?
    - A number that is the smallest Integer that is multiple of ***n*** and ***m***:
        - greater than ***n*** and ***m*** and smaller than or equal to ***n*** * ***m***

```
n = int(input("Enter a number: "))
m = int(input("Enter another number: "))
if (n < m):
    initial = m
else:
    initial = n
for i in range(initial, (n*m)+1):
    if ((i % n) == 0) and ((i % m) == 0):
        print("The LCM between", n, "and", m, "is", i)
        break
```

Try it for 6 and 8, then for 7 and 9, then for 17 and 81, and then for 567 and 1344.

... and there is always room for improvement!

Text: LCM - Least Common Multiple.

Number systems

How many fingers do you have?

What is the difference between a number and its written form?

We will briefly see how to read numbers using different number systems, and because we are in Computer Science, we will pay further attention to binary and its operations, up to logarithm:

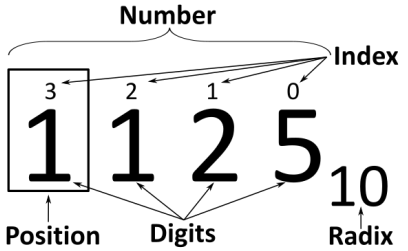
- **Decimal and Binary**
  - Binary logarithms
- Octal and Hexadecimal

Ideas and Expressions

- What is a dog?
  - Is it a three letter word: *d*, *o*, and *g*? Or is it an animal with four legs, ears, tail, eyes, mouth, snout, fur, and everything else we associate with the idea of a dog?
  - A specific dog is a very palpable thing, but the idea of a dog, not so much…
- With numbers this is also true, even a little more than for dogs, since numbers are pure abstract things.
  - Four dogs are palpable, but what about the number four itself?
    - What is the number four?

Wikipedia: The Number 4.

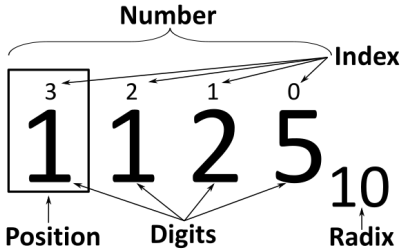
Decimal Number System



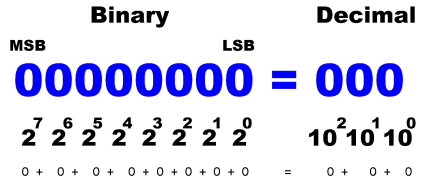
- Since Leonardo Fibonacci (*Liber Abaci*, 1202, Pisa) the Western world adopted the Arabic numbers documented by Al-Khwarizmi (Bagda, 820)
  - A decimal representation of numbers using digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
  - 4 means four (4 times 1)
    - 4 times 10^0
  - 14 means fourteen (1 times 10 plus 4 times 1)
    - 1 times 10^1 plus 4 times 10^0
  - 41 means forty one(4 times 10 plus 1 times 1)
    - 4 times 10^1 plus 1 times 10^0
  - 423 means four hundred twenty three
    - 4 times 10^2 plus 2 times 10^1 plus 3 times 10^0

Wikipedia: Decimal.

Positional Numeral System



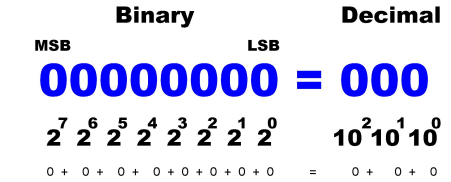
- The most common one is the **decimal** system that uses the number **ten** as radix, with digits from **0** to **9**.
  - These are the numbers you've known for your whole life.
- But what if the radix is number two, with digits **0** and **1**?
  - (a bit is a amount of information holding 0 or 1)
- This is the **binary** system, extensively used in computers:
  - 100 means four because it is:
    - 1 times 2^2 plus 0 times 2^1 plus 0 times 2^0
  - 1101 means thirteen because it is:
    - 1 times 2^3 plus 1 times 2^2 plus 0 times 2^1 plus 1 times 2^0

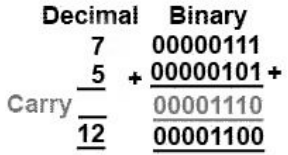


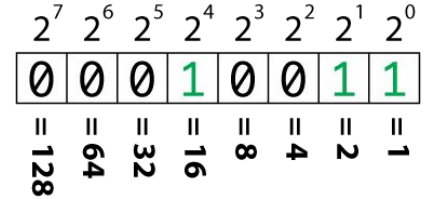
Wikipedia: Positional notation.

The Binary System

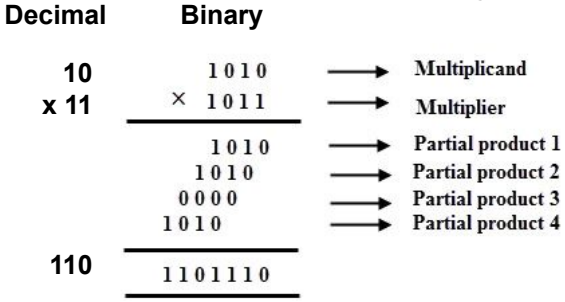
- The basic operations on binary numbers are the same as in any system:
  - Sum:
    - 0 + 0 = 0
    - 0 + 1 = 1
    - 1 + 1 = 10
    - 1 + 1 + 1 = 11
  - Multiplication:
    - 1 * 0 = 0
    - 1 * 1 = 1

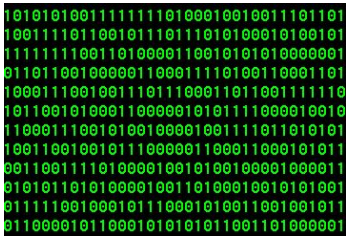






16 + 2+1 = 19





Text: Binary Number System.

The Binary System

- In Python you can have a number expressed in binary using binary literals:
  - ***0b10*** or ***0B10*** are both the number two expressed in binary;
  - the built-in function ***bin(n)*** converts the number ***n*** into a *string* holding the prefix ***0b*** and the bits (***0*** and ***1***'s) of binary representation of ***n***.

Try it!

```
a = 0b10      # which is two
b = 0b1100    # which is twelve
c = 0B101     # which is five

print("This should be two:", a)
print("This should be twenty four:", a * b)
print("This should be seven:", a + c)
print("This should be fifty:", c * 10)

twoInBinary = bin(2)
twelveInBinary = bin(12)
fiveInBinary = bin(5)

print()
print("This should be two in binary with preambule:", twoInBinary)
print("This should be twelve in binary with preambule:", twelveInBinary)
print("This should be five in binary with preambule:", fiveInBinary)

print()
print("This should be two in binary:", twoInBinary[2:])
print("This should be twelve in binary:", twelveInBinary[2:])
print("This should be five in binary:", fiveInBinary[2:])
```

There are 10 types of people in this world, those who understand binary and those who don't!

Wikipedia: Binary Numbers.

Number systems

How many fingers do you have?

What is the difference between a number and its written form?

We will briefly see how to read numbers using different number systems, and because we are in Computer Science, we will pay further attention to binary and its operations, up to logarithm:

- Decimal and Binary
  - **Binary logarithms**
- Octal and Hexadecimal

Powers of 2

Knowing the powers of two is the hallmark of Computer Scientists, and for many reasons.

One of the reasons is to refer to storage size.

Another reason is because many algorithmic decisions rely on splitting problems in two, which leads us to binary logarithms.

A future computer scientist must know powers of 2.

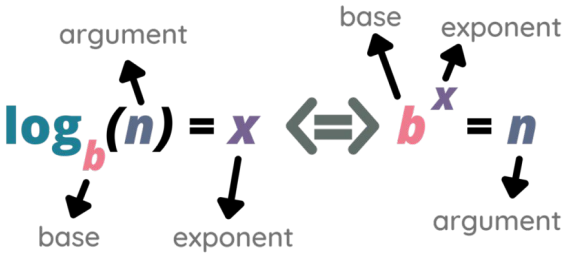
Logarithms

In mathematics every operation has its opposite:

- Sums are the opposite of subtraction;
- Multiplications are the opposite of division;
- Power is like the opposite of logarithms.

If a number ***b*** power a number ***x*** is equal to ***n***, then the logarithm of ***n*** base ***b*** is ***x***.

- $\log_{10}(1000) = 3$ (decimal or common log)
- $\log_{2}(1024) = 10$ (binary log)
- $\log_{2.718281}(100) = 4.60517$ (natural log)
- $\log_{5}(15625) = 6$ (logarithm base 5)



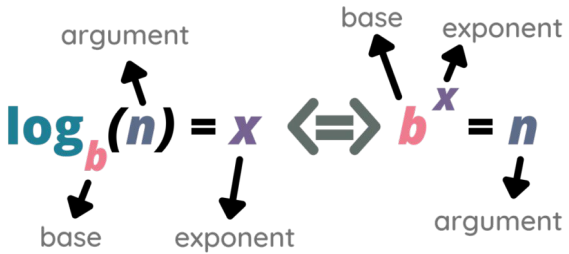
One particular comfortable logarithm for numeric handling is the one with base ***e*** (Euler's number) which is a non rational number equal to 2.718281...

One can use this logarithm property to find logarithms of any base:

$$\log_b x = \frac{\log_k x}{\log_k b}$$

Wikipedia: Logarithm.

Logarithms



In Python you can use the ***math*** module which has the functions ***log(n)***, ***log2(n)***, and ***log10(n)*** to compute the natural, binary and decimal (common) logarithms.

The ***math*** module has other cool stuff about mathematics, like some irrational numbers constants like pi (**π**) and Euler's number (**e**).

$$\log_b x = \frac{\log_k x}{\log_k b}$$

```
import math

print(math.pi)
print(math.e)

print()
print("common log of 1000:", math.log10(1000))
print("binary log of 1024:", math.log2(1024))
print("natural log of 100:", math.log(100))

a = math.log(15625) / math.log(5)
print("log base 5 of 15625:", a)
```

Try it!

Text: The math module.

Number systems

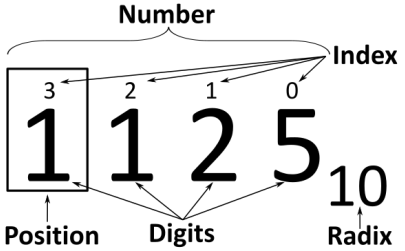
How many fingers do you have?

What is the difference between a number and its written form?

We will briefly see how to read numbers using different number systems, and because we are in Computer Science, we will pay further attention to binary and its operations, up to logarithm:

- Decimal and Binary
  - Binary logarithms
- **Octal and Hexadecimal**

The Other Number Systems



In Computer Science, other number systems with bases that are powers of two are frequently employed as a way to summarize strings of bits.

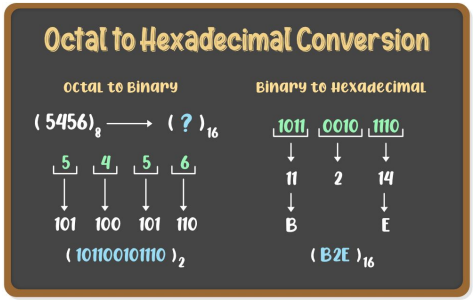
- Octal system, representing 3 bits:
  - base: 8
  - digits: 0, 1, 2, 3, 4, 5, 6, and 7
- Hexadecimal (Hex) system, representing 4 bits:
  - base: 16
  - digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F

- eight in Octal: 10
- fifteen in Octal: 17
- eighty in Octal: 120

- twelve in Hex: C
- fifteen in Hex: F
- sixteen in Hex: 10
- eighty in Hex: 50

Text: Converting Octal to Hex (code beautify) or Converting Octal to Hex (rapid tables).

Octal and Hex to Binary



Because Octal and Hex were made to summarize strings of bits, which are actually binary, the conversion towards binary representation is pretty straightforward:

- for Octal, each octal digit becomes three bits ($8 = 2^3$)
  - Octal `10` becomes: `001 000`
  - Octal `17` becomes: `001 111`
  - Octal `120` becomes: `001 010 000`
- for Hex, each hex digit becomes four bits ($16 = 2^4$)
  - Hex `C` becomes: `1100`
  - Hex `F` becomes: `1111`
  - Hex `10` becomes: `0001 0000`
  - Hex `50` becomes: `0101 0000`

- eight in Octal: 10
- fifteen in Octal: 17
- eighty in Octal: 120

- twelve in Hex: C
- fifteen in Hex: F
- sixteen in Hex: 10
- eighty in Hex: 50

Video: Positional Number Systems.

Octal and Hex in Python

- In Python you can express a number in octal and hex using literals:
  - ***0o10*** or ***0O10*** are both the number eight expressed in octal;
  - ***0x10*** or ***0X10*** are both the number sixteen expressed in hex;
  - the built-in functions ***oct(n)*** converts the number ***n*** into a *string* holding the prefix ***0o*** and the digits of octal representation of ***n***;
  - the built-in functions ***hex(n)*** converts the number ***n*** into a *string* holding the prefix ***0x*** and the digits of hex representation of ***n***;

Try it!

```
print("This should be 16 / 16:", a / b)
print("This should be 74 + 16", b + c)

twelveInOctal = bin(12)
twoHundredFiftyFiveInHex = hex(255)
print()
print("This should be twelve in octal with preambule:", twelveInOctal)
print("This should be two hundred fifty five in Hex:", twoHundredFiftyFiveInHex[2:])
```

Text: Binary, Hex, and Octal in Python.

This Week's tasks

- Post discussion D#2
- Coding Project P#2
- Quiz Q#2

Tasks

- Post in the discussion how strange was the number systems to you, and which of the topics seen today was more surprising to you.
- Coding Project #2, converting bases.
- Quiz #2 about this week topics.

Post Discussion - Week 2 - D#2

Post in the discussion how strange were the number systems to you, and which of the topics seen today was more surprising to you.

- You might be aware that there were different number systems, but does it mean that you can reason in numbers coded in different bases?
- 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 discussion in the message board by this Monday;
- Reply to posts of your colleagues in the message board by next Thursday.

This task counts towards the Discussions grade.

Second Coding Project - Week 2 - P#2

- Write a Python program that:
  - Asks the user a number as a string (it's ok to set a max number of digits);
  - Asks the user for the base (an Integer from 2 to 16);
  - With that information, your program should compute the binary and decimal representations of the number, then print it out as a string.
  - For example, if the user enters: **FA** and **16**, your program should print out:
    - "**FA in base 16 is: 250 in base 10 and 11111010 in base 2**"
  - If the user enters: **34** and **5**, your program should print:
    - "**34 in base 5 is: 19 in base 10 and 10011 in base 2**"

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.

Second Quiz - Week 2 - Q#2

- The second quiz in this course covers the topics of Week 2.
- The quiz will be available this Saturday, and it is composed of 10 questions.
- The quiz should be taken on Canvas (Module 2), 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.
- However, 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 2 of CSC 6000

- Post discussion D#2 by Monday, reply to colleagues by next Thursday;
- Do quiz Q#2 (available Saturday) by next Thursday;
- Develop coding project P#2 by next Thursday.

Next Week - Arithmetic and Geometric Progressions, Summations

MERRIMACK COLLEGE