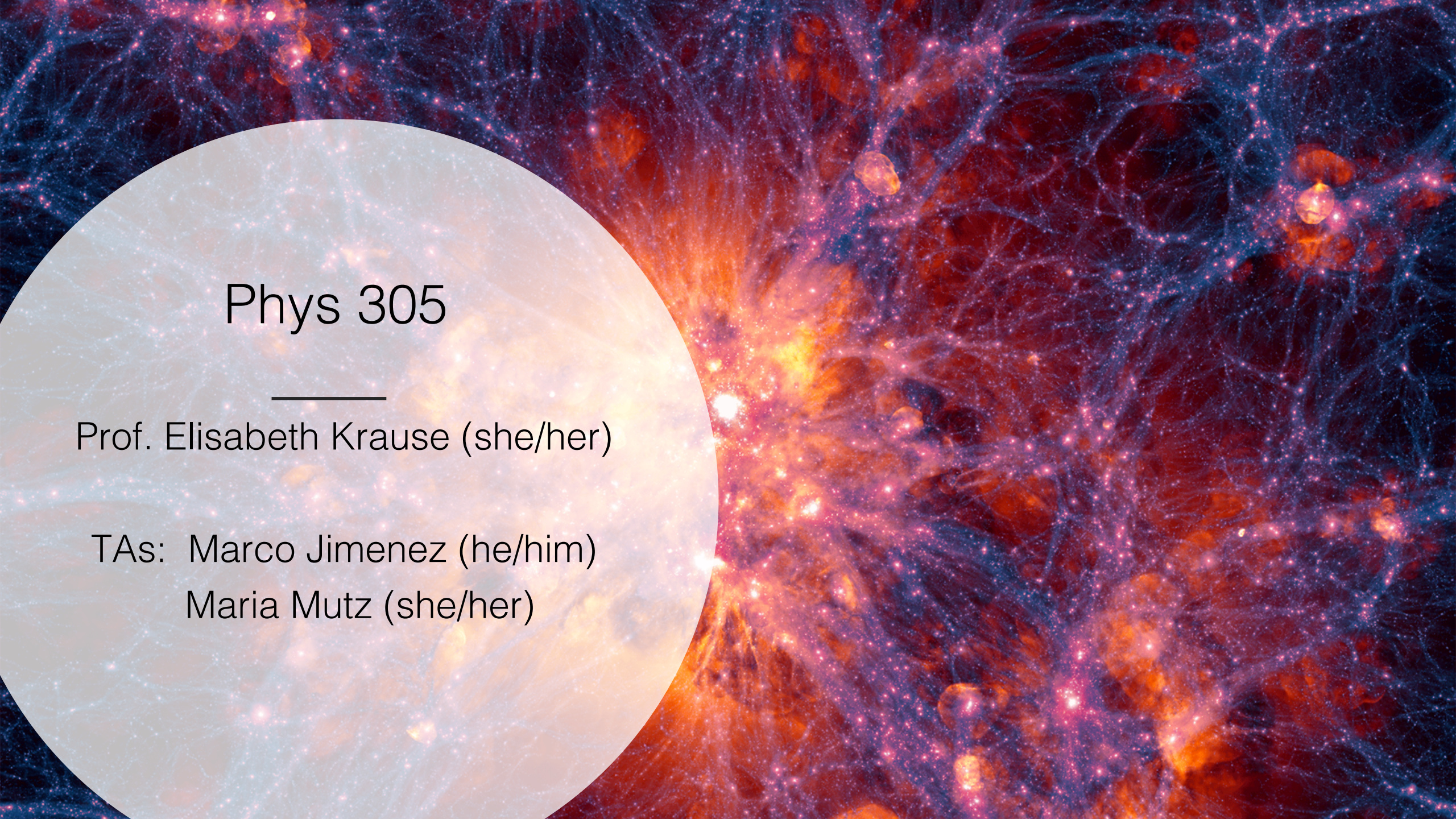

Phys 305

Prof. Elisabeth Krause (she/her)

TAs: Marco Jimenez (he/him)
Maria Mutz (she/her)

Today's Lecture

- Algorithms and computer programs
- Basic python syntax: operators, loops, functions, recursion
  - python intro in Prof. Pinto's GuerrillaGuide.pdf on D2L (-> Course Content)
- Running python: command line, (jupyter) notebooks
- Algorithmic thinking break-out:
  - Prime factorization

Algorithms & Computer programs

- *Algorithm*: a set of instructions
  - e.g., the equations governing the gravitational interactions of two particles
- *Computer program*: executes a set of algorithms *expressed in a programming language*

- There are many different programming languages, with different strengths/weaknesses and different levels of abstraction
  - high level languages: “programmer friendly” - relatively close to human language, programming (mostly) does not require detailed knowledge of hardware
  - low level languages: “machine friendly” – relate to specific architecture, typically hard to read, programming takes much longer

Algorithm: backing up a car (high-level)

- Are there two or three pedals?

  If two pedals (automatic transmission):
  - Press down the left pedal.
  - Put the key into in the ignition.
  - Turn the key until the car starts, then let go.
  - Move the shift lever to “reverse”.
  - Slowly let go of the left pedal.
  - …

  If three pedals (automatic transmission):

Algorithm: backing up a car (low-level)

- Are there two or three pedals?

  If two pedals (automatic transmission):
  - Grab the wide end of the key.
  - Insert the pointed end of the key into the slot on the lower right side of the steering column.
  - Press down the left pedal.
  - Rotate the key about its long axis in the clockwise direction (when viewed from the wide end toward the pointed end).
- …

You're in luck - this class will use python, a very high-level language!

Memory, variables, and expressions

- Computer memory: ~enormous list of binary digits.

  need to know the **location** of any given datum and its **contents**

  Low-level: refer to every location in memory by its numerical address ("add the number stored at location 8283747723 to the number stored at 4239128797 and put the result in location 7512398794")

  high-level programming languags simplify the bookkeeping by providing a mnemonic name for a location in memory: a **variable**.

  When we say in Python `var = 4`, python chooses an unused location in memory and associates it with the text `var` – much easier for us to remember.

  var then refers to the contents of that location, wherever it may be.

Memory, variables, and expressions

- A *program* is a set of instructions for the computer to take some action.

  If we write `dummy = var`, this is not the mathematical statement that `dummy` is equal to `var`. Instead, the statement will be translated by the compiler into instructions for the CPU which say, roughly:

  1. Take the datum stored in the location in memory associated with `var` and stick it in a temporary location within the CPU (known as a register).

  2. Take the contents of that register, and store it in the location in memory associated with `dummy`.

- The symbol **=** in the above statement is an **operator**;

  specifically, it is the assignment operator, assigning the value appearing to its right to the variable appearing on its left. The expression to the right can be more complex. We can for example write `x = var + 24`.

  What might this statement be to the computer?

Variables and operators

- Variable: symbolic name associated with storage location in computer memory
  - any variable has a **location** and **value**
  - Python variables can have different **types**, i.e. hold data in different formats
  - Describe the data types you encountered in Lecture 2
    - Integer: hold whole numbers
    - Float: hold real numbers
    - Boolean: hold one of the two values True, False
- Operator: perform operation on variable/value on its right
  - X = 3: = assigns variable X the value 3
  - Not True: Not returns the opposite boolean value
    - turns true to false and false to true

Boolean operators

g BOOLEAN_OPERATOR h

- the NOT operator ... Returns the opposite boolean value
  - turns true to false and false to true
- the AND operator ... Returns true only if both g and h to be true, else returns false
- the OR operator ... Returns true if either g or h is true. Will return false if g and h are both false

Conditional Statements

- Boolean Operators and Comparison Operators can be used to build more interesting expressions
- A **conditional statement** executes if a given expression is true. The general syntax is

```
if EXPRESSION:
        STATEMENT
elif EXPRESSION2:
        STATEMENT2
else:
        DEFAULT_STATEMENT
```

Loops

- a **loop** is a sequence of instructions that is continually repeated until a condition is reached
- Today, we'll cover two types of loops:

While Loops repeat as long as a certain boolean condition is still true.
The general syntax is :

```
while BOOLEAN_EXPRESSION:
        STATEMENT
```

For Loops iterate a given number of times.
The general syntax is:

```
for RANGE:
        STATEMENT
```

Functions

- A **function** is a block of code which only runs when it is called. You can pass data, known as parameters, into a **function**. A **function** can return data as a result.
- This is a convenient way to write code that can be reused without copying.
- In python, functions are defined using the block keyword "def", followed with the function's name as the block's name. The instructions within the function block are indented.

```
def my_function (<parameters>):
        STATEMENT
        return <result>
```

Recursion

- **Recursion** is a method of programming or coding a problem, in which a **function** calls itself one or more times in its body. Usually, it is returning the return value of this **function** call.
- Example: Factorial function (!): n! = 1*2*…*(n-1)*n

```
def factorial(n):
        if n == 1:
                return 1
        else:
                return n * factorial(n-1)
```

Running python

- **Command line**
  - In the terminal window:
  - *run python on the command line*
  - **$** python
  - *use python as a calculator*
  - >>> 2+3
  - >>> exit()
  - *Execute python program test.py*
  - **$** python test.py

Running python

- **Notebook**
  - Start a jupyter notebook session (or your preferred notebook interface)
  - *run python on the command line*
  - **$** jupyter notebook
  - Edit notebook in your browser

Example: Recursion

```
#!/usr/bin/env python

def factorial(n):
    f = 1
    for i in range(2,n+1):
        f = f * i
    return f

print(factorial(4))
```

Example: Argument Input

```
#!/usr/bin/env python

import sys

nargs = len(sys.argv)

if nargs == 1:
    print("no args")
else:
    print(nargs-1,"args")
    for i in range(1,nargs):
        print("arg {} is {}".format(i,sys.argv[i]))
```

Your Turn: Prime Factorization

Write a program which takes an integer input and writes out its prime factors.

- 1) determine algorithm (Don't write code!!)
- 2) Sketch out code
- 3) Fill in code details

We will split up into break-out rooms.

Each room designates a scripe, who will share their screen.

Jointly develop the algorithm and code.

Maria, Marco and I will circle through breakout rooms.

You can leave before end of class if you group finishes early.