Discrete Structures!

CMPSC 102

The Great Review

ALLEGHENY COLLEGE

Course Description

CMPSC-102 Discrete Structures (4 Credits) An introduction to the foundations of computer science with an emphasis on understanding the abstract structures used to represent discrete objects. Participating in hands-on activities that often require teamwork, students learn the computational methods and logical principles that they need to create and manipulate discrete objects in a programming environment. Students also learn how to write, organize, and document a program's source code so that it is easily accessible to intended users of varied backgrounds. During a weekly laboratory session students use state-of-the-art technology to complete projects, reporting on their results through both written documents and oral presentations. Students are invited to use their own departmentally approved laptop in this course; a limited number of laptops are available for use during class and lab sessions.

Key Questions

- How do I connect mathematical terminology (i.e,. mapping, function, number, sequence, and set), to the implementation of Python programs that declare and call functions and declare and manipulate variables?

- How do I use iteration and conditional logic in a Python program to perform computational tasks like processing a file's contents and mathematical tasks like using Newton's method to approximate the square root of a number?

Key Questions

- How do I use non-recursive functions, recursive functions, and lambda expressions to perform mathematical operations such as computing the absolute value of a number and the means of a sequence of numbers?

- How do I use virtual environments like Venv, Poetry, Typer and other resources to create a professional project?

Key Questions

- How do I use the mathematical concepts of ordered pairs, n-tuples, lists and dictionaries to implement functions with a clearly specified behaviors?

- How do I employ the mathematical concepts of sequences, monoids, and lists to implement efficient Python programs that use functions with a clearly specified behavior to perform tasks like finding a name in a file or computing the arithmetic mean of data values?

Key Questions

- How do I use dynamically generated streams of data to implement memory efficient and predictable Python programs?
- How do I use the mathematical concepts of sets and Boolean logic to design Python programs that are easier to implement and understand?

Key Questions

- How do I implement finite sets in Python so that I can calculate and use probabilities?
- How do I implement data structures to create plots? How do I install such masterful software to do this?!

Key Questions

- How can I create basic statistics from text and then explain my results using values and plots?
- How can I describe data using statistical tools such as correlation, variance, standard deviation and others?

Key Questions and Learning Objectives

- How do I connect mathematical terminology (i.e,. mapping, function, number, sequence, and set), to the implementation of Python programs that declare and call functions and declare and manipulate variables?

- To remember and understand some discrete mathematics and python programming concepts, setting the stage for the exploration of Discrete Structures.

Discrete Structures - In terms of programming

Discrete Structures = Math + Code

Discrete mathematics is composed of fundamental concepts such as:

- Symbols, character strings, and truth values.
- Objects and collections of these entities (e.g., stored in sets or tuples).

Specification (S) and Program (P)

- Specification (S): The detailed description and design of a computer program.
  - Define the input, output, and internal objects.
  - Use the vocabulary of discrete mathematics to formalize ideas.
- Program (P): The implementation of the specification in a programming language.
  - Translate the specification into code.
  - Implement and test the program.

Our goal: To implement a program P that meets a particular specification S

Finding Solutions

How do we think about our programming?

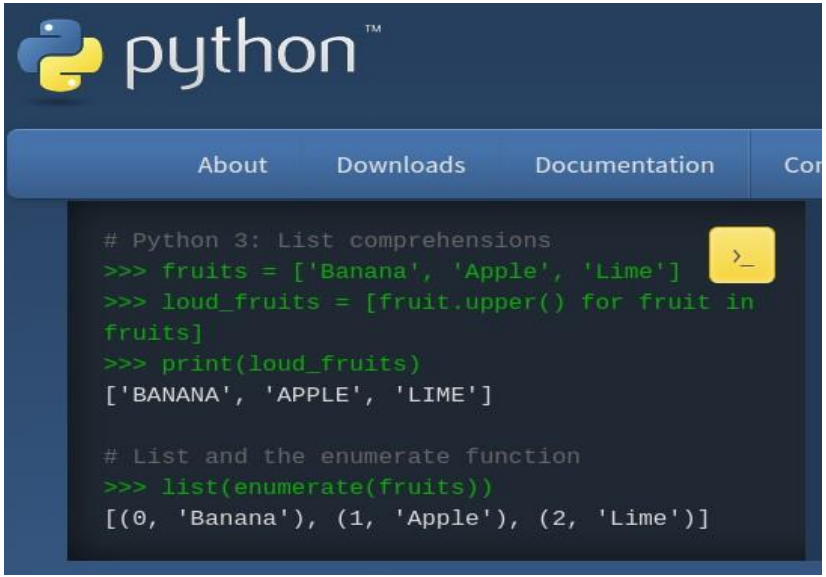
- To find solutions, we frequently jump from a discrete mathematical specification to a Python program and back again to the specification to prepare a software solution to the problem.
- Pick the suitable level of abstraction for the problem you solve (*and the solution soon presents itself!*)

Discrete Structures with Python

Python

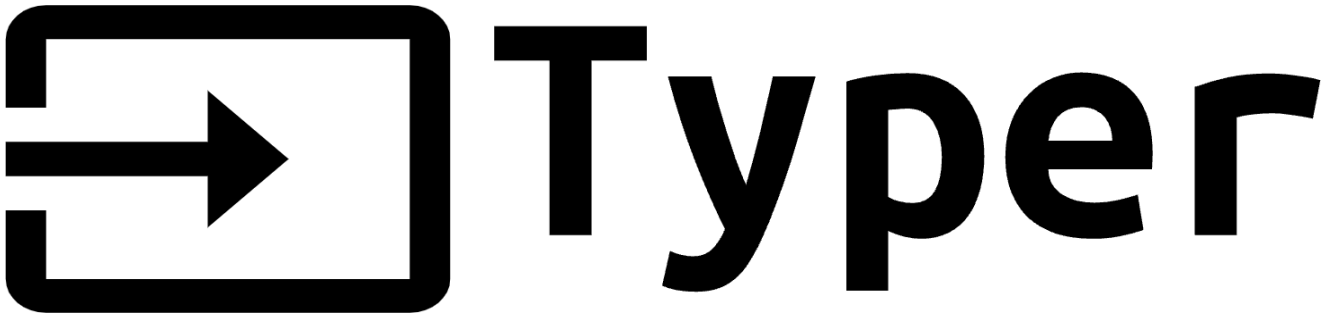
- Discrete structures support precise programming
- Benefits of using Python to explore discrete structures
- Modern language with exceptional package support
- Clean syntax and semantics that is easy to learn
- Out-of-the-box support for many discrete structures
- The semantics of the language match those of discrete structures (the programming language resembles the mathematics that you might employ in your work!)

Get Python3



- Get Python3 from the Python Software Foundation: https://www.python.org/downloads/

Python Resource - Typer



https://typer.tiangolo.com/

- Command line interface support for program inputs and parameters
- Annotations: assigns types to functions that accept arguments (parameters)
- Productivity: types aid in the creation of the interface
- Checking: Confirm that inputs match expected types.

Python Resource - Poetry

https://python-poetry.org/

- Management support for Python and its resources
- Environments: manage dependencies in isolation
- Package: create a stand-alone executable application
- Publish: expedite and simplify the release of program to PyPI

Key Components

All programs built out of

- **Function calls**: Granting temporary kernel-time and/or using issuing parameters to a sub-sequence of instruction in a program.
- **Assignment statements**: The issuing of a value to a variable or place in memory to contain the value.
- **Iteration constructs**: Structures used in computer programming to repeat the same computer code multiple times (*loops*).
- **Conditional logic**: the use of logical rules in code to govern steps taken.
- **Variable creation**: The introduction of an object in memory to contain some value.
- **Variable computations**: The use of values contained in variables to create new value using an operator.
- **Variable output**: The revealing of some value in a variable by printing or another means.

Practical Variable Limitations in Python

More computational limits

Python Output:

```
>>> 1.0 == 1.1
False
>>> 1.0 == 1
True
>>> 'h' + 'i' + '!'
'hi!'
>>> .33333 + .33333 + .33333 == 1
False
>>> .33333333333 + .33333333333 + .3333333333 == 1
False
>>> 1/3
0.3333333333333333
>>> 1/3 + 1/3 + 1/3 == 1
True
```

Key Questions and Learning Objectives

- How do I use **iteration** and **conditional logic** in a Python program to perform computational tasks like processing a file's contents and mathematical tasks like using Newton's method to approximate the square root of a number?

- To remember and understand some discrete mathematics and Python programming concepts, setting the stage for exploring discrete structures.

A program is a sequence of statements

To be philosophical for a moment …

```
file = open("emails")
for line in file:
        name, email = line.split(",")
        if name == "John Davis":
                print(email)
```

- A Python program is a sequence of statements about mixing things with the rest of the ingredients … like a recipe
- There is a list of ingredients
- There is a sequence of events about when to use each ingredient Timing (run time) is important
- (Chef, waiter, guests) == (programmers, instructions, users)

Quadratic Root Calculation

Quadratic Equation:

$$ax^2 + bx + c = 0 \quad (2)$$

Quadratic Formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (3)$$

Special Note

Note the $x_{1,2}$ to imply that there are two solutions (i.e., x_1 and x_2) to find for a second-degree equation as observed from the x^2.

Quadratic Roots – The Problem Defined

To Solve: $x^2 + 3x - 4 = 0 \quad (1)$

Want to have roots

$$x_1 = ? \; and \; x_2 = ?$$

Key Questions and Learning Objectives

- How do I use non-recursive functions, recursive functions, and lambda expressions to perform mathematical operations such as computing the absolute value of a number and the means of a sequence of numbers?

- How do I use virtual environments like Venv, Poetry, Typer and other resources to create a professional project?

Absolute Value of a Number

A function to calculate value

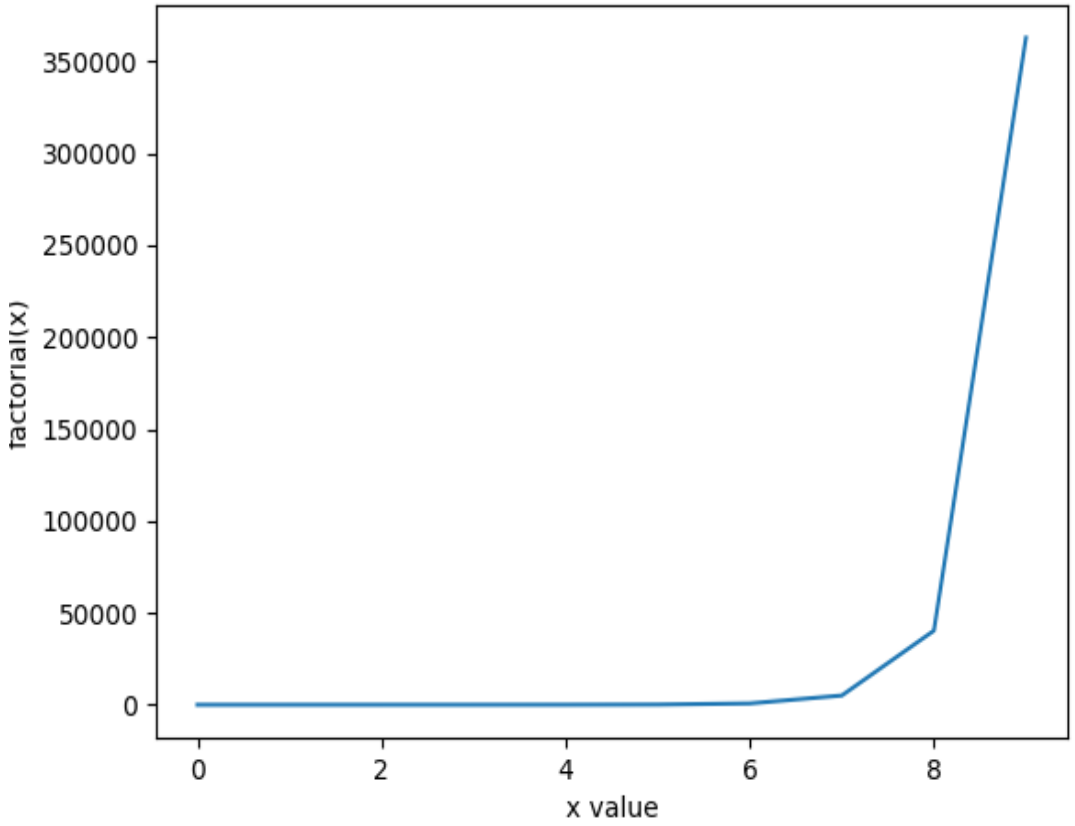
Function for finding absolute values

```
def abs(n):
    if n >= 0:
        return n else:
        return -n
```

Speaking Pythonically

- What is the meaning of the operator >= ?
- What is the output of `print(str(abs(10)))` ?
- What is the output of `print(str(abs(-10)))` ?
- Are there other ways to implement this function ?

Factorials - values get quickly get big



x	*fac*(*x*)
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800

Factorials

Factorials: one definition

$$N! = \pi_{i=1}^{N} i = 1 * 2 * .. * (N-1) * N$$

Factorials: another definition

$$N! = \frac{(N+1)!}{(N+1)} = \frac{(N+1) * N!}{(N+1)}$$

Factorials are applied to integers

Factorials

Factorials

$$N! = N * (N-1) * (N-2) * \cdots * (2) * (1)$$

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

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

$$3! = 3 * 2 * 1$$

$$2! = 2 * 1$$

$$1! = 1$$

$0! = 1$ (Special case by convention)

Factorials defined

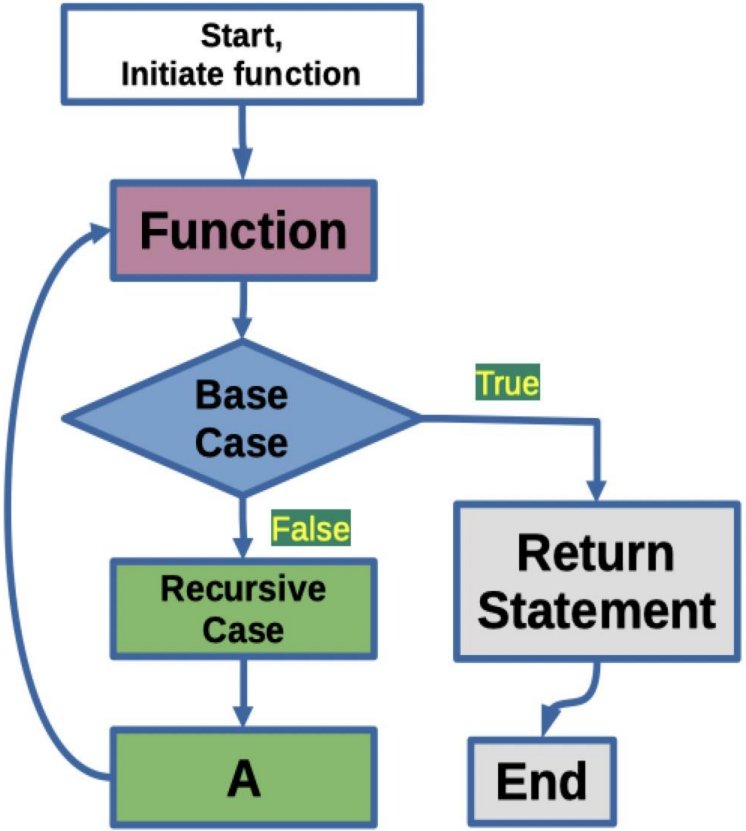
$$N! = [(N-1)! + (N-2)!] * (N-1)$$

$$7! = (6! + 5!) * 6$$

$$6! = (5! + 4!) * 5$$

$$5! = (4! + 3!) * 4$$

Creating Solutions



Calculating Factorials by Recursion

```
def factorial(number: int):
        if number == 1:
                return 1
        return number * factorial(number - 1)

num = 5
print("The factorial of " + str(num)
        + " is " + str(factorial(num)))
```

- The recursive *factorial* function calls itself!
- How does this function ever stop executing?
- What are the benefits to using recursive functions?

Calculating Factorials by Recursion

```
def factorial(number: int):
        if number == 1:
                return 1
        return number * factorial(number - 1)

num = 5
print("The factorial of " + str(num) +
        " is " + str(factorial(num)))
```

- Where is the base case?
- Where is the recursive case?
- How could this code work without these two functions?

Lambda Expressions - Also known as, “anonymous functions”

```
def call_twice(f, number: int):
                print(f"Calling twice {f} with number {number}")
                return f(f(number))

square = lambda x: x*x
number = 5
result = call_twice(square, number)
print("Calling square lambda twice " +
                "with " + str(number) +
                " is " + str(result))
```

- Functions are values in the Python programming language
- square is an expression that has a function as its value

Lambda Expressions

```
def call_twice(f, number: int):
    print(f"Calling twice {f} with number {number}")
    return f(f(number))
```

```
square = lambda x: x*x
number = 5
result = call_twice(square, number)
print("Calling square lambda twice " +
      "with "  +  str(number) +
      " is " + str(result))
```

```
Calling twice <function <lambda> at 0x37500c8> with number 5
Calling square lambda twice with 5 is 625
```

- Lambda functions are known as anonymous functions and add simplicity in programming
- Useful for small function input to other functions

Key Questions and Learning Objectives

- How do I use virtual environments like Venv, Poetry, Typer and other resources to create a professional project?

- To learn how to use libraries and dependencies for development with Python code and programming techniques to create the foundations for a professional project.

Setup Steps

- Make a working directory

```
mkdir projects
cd projects
```

- Use Poetry to create new project

```
poetry new hello_user
cd hello_user
```

- Add Project Dependencies

```
poetry add typer
poetry add rich
```

- Add Project Development Dependencies

```
poetry add -D black mypy
```

Mypy: http://mypy-lang.org/

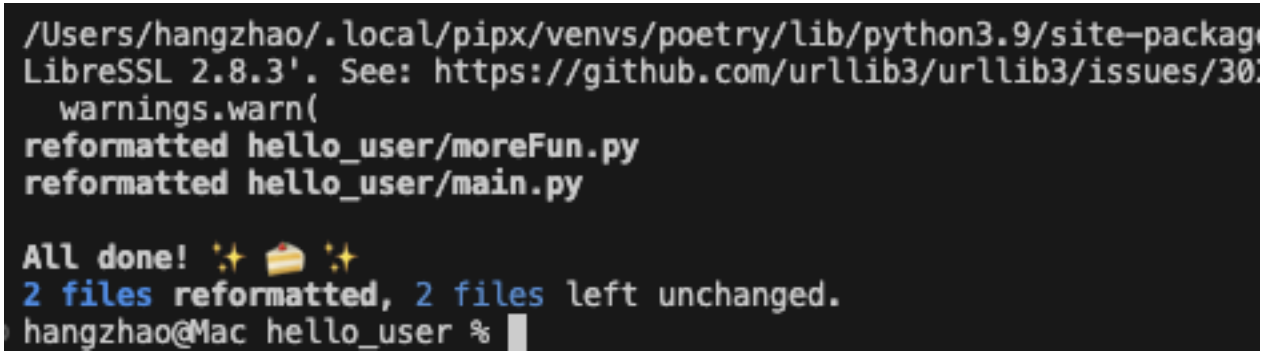
Add File: projects/hello user/hello user/main.py - File located in sanbox: main.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from rich.console import Console
import typer

# create a Typer object to support the command-line interface
cli = typer.Typer()
@cli.command()
def main(first: str = "", middle: str = "", last: str = ""):
                """Say hello to the person having a name of first, middle and last name"""
                console = Console()
                console.print(" Hello to;")
                console.print(f"\t First = {first}")
                console.print(f"\t Middle = {middle}")
                console.print(f"\t Last = {last}")
# end of main()
```

Basic Reformatting with Black

poetry run black hello_user tests



Execute Project

What do you see?

```
# run from projects/hello_user/hello_user
poetry run python3 hello_user/main.py --help
```

Key Questions and Learning Objectives

- How do I use the mathematical concepts of ordered pairs, n-tuples, lists and dictionaries to implement functions with a clearly specified behaviors?

- To remember and understand some discrete mathematics and Python programming concepts, enabling the investigation of practical applications

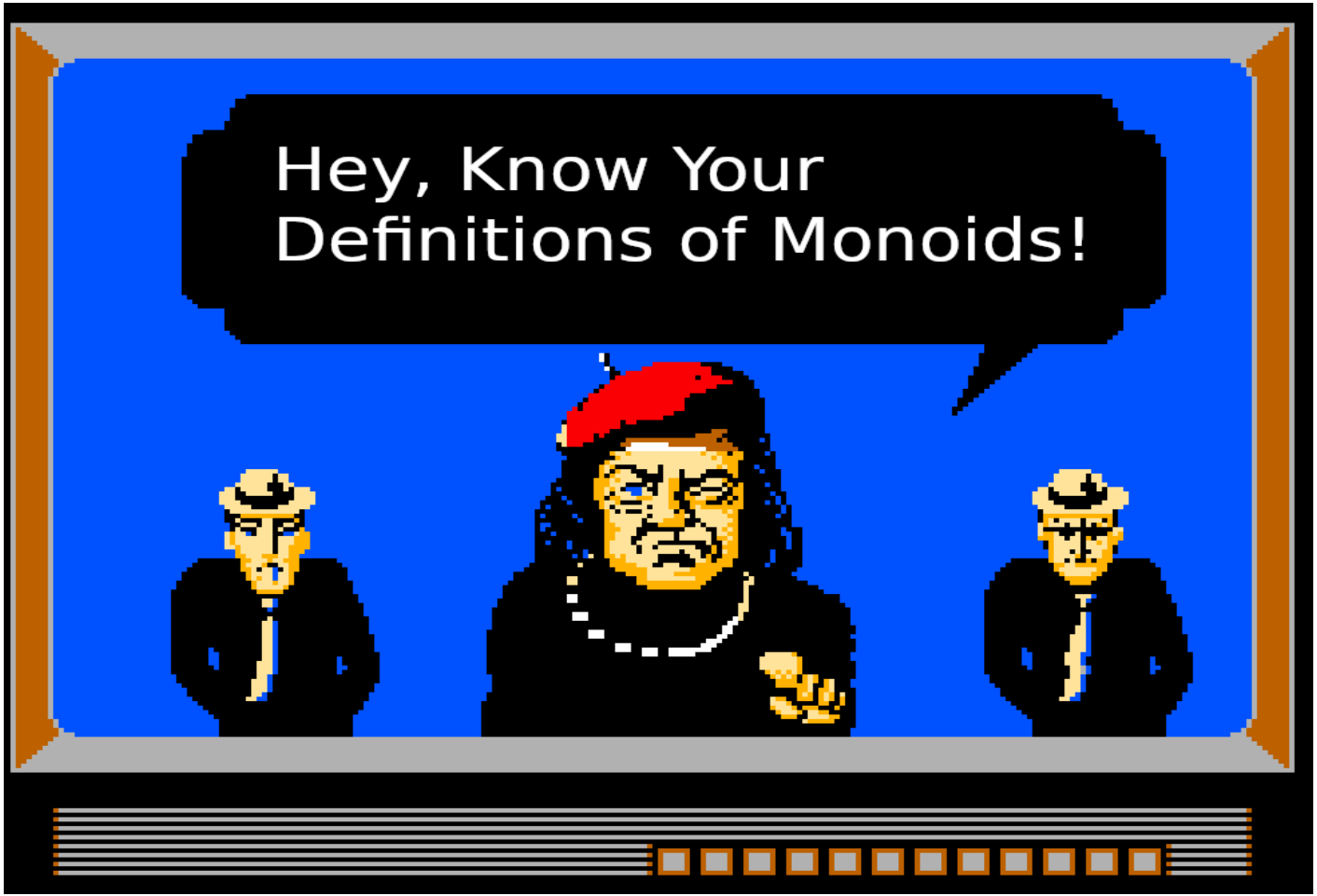
What are Ordered Pairs?

- Mathematical concepts yield predictable programs
- Understanding the concept of an ordered pair:
    - **Pair**: a grouping of two entities
    - **Ordered**: an order of entities matters
    - **Ordered Pair**: a grouping of two entities for which order matters
    - **Coordinate on Earth**: the latitude and longitude coordinates are an ordered pair
    - **Complex Numbers**: the real and imaginary parts are an ordered pair
    - An ordered pair is not the same as a set of two elements! Why?
    - Can we generalize to an ordered grouping beyond two entities? How?

Key Questions and Learning Objectives

- How do I employ the mathematical concepts of sequences, monoids, and lists to implement efficient Python programs that use functions with a clearly specified behavior to perform tasks like finding a name in a file or computing the arithmetic mean of data values?

- To remember and understand some the concept of a monoid, seeing how it connects to practical applications with strings and sequences


Hey, Know Your
Definitions of Monoids!

Monoid Definition

- In Abstract Algebra, a monoid is a **set** equipped with an **associative binary operation** and an **identity element**. For example, the non-negative integers with addition form a monoid, the identity element being 0.

- A monoid is a combination of an object (a,b,c) and an operation (+) that meets the following conditions
  - the operation on two of the objects produces a new object of the same kind
    - int + int = int
  - associative operations
    - (a+b) + c = a + (b+c)
  - a null object e must exist, such that $e + a = a + e = a$
    - $n + 0 = n$

Ref: https://en.wikipedia.org/wiki/Monoid#Definition

Higher-Order Sequence Functions

- Functions that work for **any sequence**?
- These **Higher Order** functions should work for lists, ordered pairs, tuples:
    - map: Apply a function to every element of a sequence
    - filter: Apply a boolean function to every element of a sequence, returning only those matching the filter's rules
    - reduce: Apply a function that acts like a binary operator to a sequence of values, combining them to a single value
- These three operators give a **vocabulary** for implementing complex, yet easy-to-ready programs in a functional programming style
- These functions are **higher-order** because they accept function as input

Key Questions and Learning Objectives

- How do I use the mathematical concepts of sets and Boolean logic to design Python programs that are easier to implement and understand?

- To remember and understand some concepts about the set, exploring how its use can simplify the implementation of programs.

General Sets

What is a set?

- For example, the numbers 1, 2, and 3 are distinct objects when considered separately, but when they are considered **collectively,** they form a single set of size three, written {1,2,3}.
- Set theory is now a ubiquitous part of mathematics,
- May be used as a foundation from which nearly all of mathematics can be derived (From 19th century mathematical thinking!)

Types of Sets

Intentional and Extensional

Question: What kind of set do we have?

Answer: We can provide two main definitions of sets.

Intentional definition of sets: *I intend this set to be …*

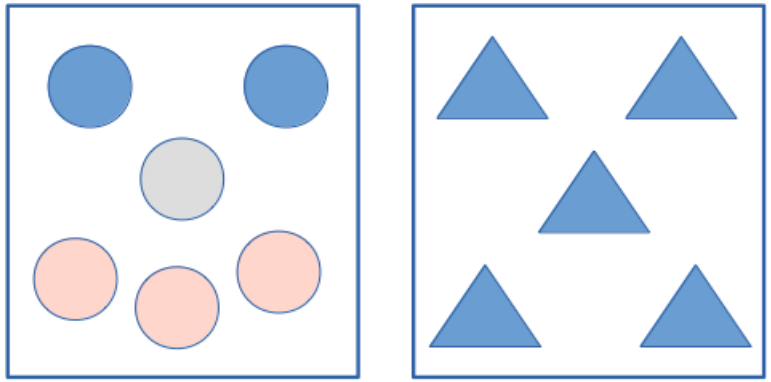
- Defines a set by specifying the necessary and sufficient conditions for when the set should be used.

Extensional definition of sets: *Logically this set is …*

- Defines a set by some definition of a concept or a term.

Types of Sets

Intentional: One decides which elements make up a set



Intentional definition of sets: *I intend that these set be ...*

- The set of blue, grey and pink circles
- The set of blue triangles
- The set of colors of the Union Jack (i.e., the British flag)

Types of Sets

Extensional: Sets of members in curly brackets

Extensional definition of sets

- $A_2 = \{4, 2, 1, 3\}$
  - The first four positive numbers
- $B_2 = \{\text{Blue, Red and White}\}$
  - The set of colors of the Union Jack (the British flag)

Types of Sets

Extensional definition of sets: a list of its members in curly brackets

- **Intentional Definition:**
  - A_1 is the set are the first four positive integers.
  - B_1 is the set of colors of the Union Jack
- **Extensional Definition:**
  - $A_2 = \{4, 2, 1, 3\}$
  - $B_2 = \{\text{Blue, Red and White}\}$

Specify a set *intentionally* or *extensionally*

- In the examples above, for instance, $A_1 = A_2$ and $B_1 = B_2$

Key Questions and Learning Objectives

- How do I place an equation and a system of logic into Python code?
- To remember and understand some the concepts involved with placing mathematical logic into code.

Rules: the Collatz or Hailstone Problem

$$f(x) = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is odd} \end{cases}$$

- The 3x+1 problem concerns an iterated function
- The question is to determine whether the function always reaches a value of 1 when starting from any positive integer.

Key Questions and Learning Objectives

- How do I implement finite sets in Python so that I can calculate and use probabilities?

- To remember and understand some concepts about sets, as implemented by SymPy, supporting the calculation of probabilities.

Mathematical Sets in Python Programs

- Set theory is useful in mathematics and computer science
- The Sympy package gives an implementation of finite sets
    - Remember, sets are "containers" for other elements
    - The sets in **Sympy** are finite sets, called **FiniteSet**
    - These sets have the same properties as built-in sets
    - **FiniteSet** has a few features not provided by **set**
    - A probability is the likelihood that an event will occur
    - We can use either **set** or **FiniteSet** to study probabilities
- Investigate probability after exploring an alternative approach to sets

Creating Sets

Import sympy

- Get into a Python instance from terminal

```
python3
```

- Creating a finite set

```
import sympy as sy

empty_set = sy.FiniteSet()
print(f"{empty_set} :: {type(empty_set)}")
# EmptySet :: <class 'sympy.sets.sets.EmptySet'>
```

- Creating a finite set

```
import sympy as sy

finite_set = sy.FiniteSet(2, 4, 6, 8, 10)
print(f"{finite_set} :: {type(finite_set)}")
# <class 'sympy.sets.sets.FiniteSet'>
```

Probability

Intersection

A die can roll prime numbers ({2, 3, 5}) or odd numbers ({1, 3, 5}). What are the chances of a die roll is both prime **AND** odd? To determine this, you calculate the probability of the **intersection** of the two event sets over all possible outcomes. E= A ∩ B= {2, 3, 5} ∩ {1, 3, 5}= {3, 5}

Probability of Event A AND Event B

```
six_sided = FiniteSet(1, 2, 3, 4, 5, 6)
roll_one = FiniteSet(2, 3, 5)
roll_two = FiniteSet(1, 3, 5)
event = roll_one.intersect(roll_two)
prob = len(event) / len(six_sided) # over all outcomes
print(prob)
```

- The 'intersect' function connects to a logical 'AND' operation
- The output of this program is 0.3333333333333333. Why?

Probability

Union

A die can roll prime numbers ({2, 3, 5}) or odd numbers ({1, 3, 5}).
What are the chances of a die roll is either prime OR odd? To determine this, you calculate the probability of the union of the two event sets over all possible outcomes. E= A ∪ B= {2, 3, 5} ∪ {1, 3, 5}= {1, 2, 3, 5}

Probability of Event A OR Event B

```
six_sided = FiniteSet(1, 2, 3, 4, 5, 6)
roll_one = FiniteSet(2, 3, 5)
roll_two = FiniteSet(1, 3, 5)
event = roll_one.union(roll_two)
prob = len(event) / len(six_sided) # over all outcomes
print(prob)
```

- The 'union' function connects to a logical 'OR' operation
- The output of this program is 0.6666666666666666. Why?

AND SO MUCH MORE!!