Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

The Great Review
Fall 2022
Week 15

The Class Websites

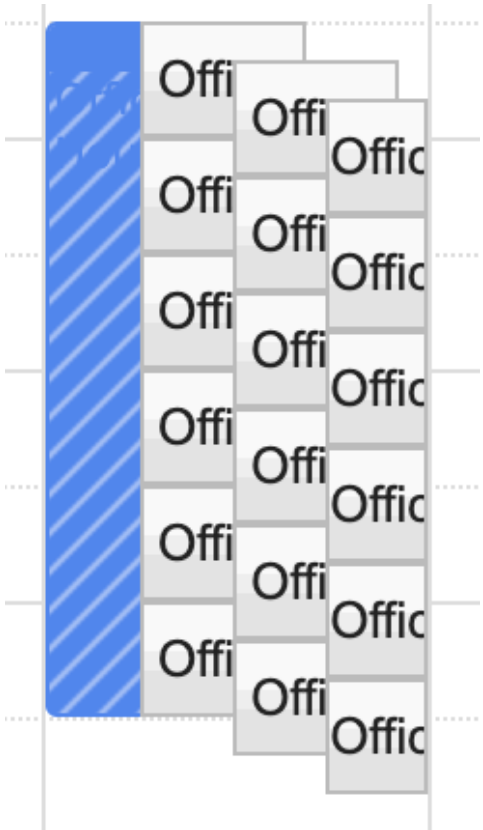
General Information

- The course Website:
  - `https://www.oliverbonhamcarter.com/classes/discretestructures/`

The Class Website

Office hours

- Booking office hours:
  - `https://www.oliverbonhamcarter.com/contactandabout/`



The Class Website

Please be familiar with the course syllabus

- Check the syllabus
  - `https://github.com/CMPSC-102-Allegheny-College-Fall-2022/classDocs/blob/main/README.md`

Figure: Did I search for *Syllabus* correctly?

Section Questions

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?

Key Questions

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?

Section Questions

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 **mean**s of a sequence of numbers?

Key Questions

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

Section Questions

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?

Key Questions

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?

Section Questions

Key Questions

How do I use **dynamically generated streams** of data to implement **memory efficient** and **predictable** Python programs?

Key Questions

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

Section Questions

Key Questions

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

Data Sets

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?

Learning Objectives

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

The Best of Both Worlds

Discrete Structures = Math + Code

- **Discrete mathematics**
  - P Made up from: *symbols*, *character strings*, *truth values*, *objects*, and *collections of these entities* as stored in *sets* or *tuples* (for example)
- S Specifying and designing a **computer program**
  - Describe input, output, and internal objects
  - Use the vocabulary of discrete mathematics
  - Implement and test the program in a language
- **Our goal**:
  - To implement a program P that meets a particular specification S

Why combine mathematics and computer programming?

Programs and Discrete Structures

Our Goal

Jump to different levels of abstraction (i.e., high-level versus low-level or mathematical versus technical) when we create programs

What **is** a computer program?

- Informal or intuitive specification
- Precise discrete mathematical specification
- Realization of a specification in Python program
- Bits packaged into bytes and words stored on a disk
- A process in execution on a CPU and stored in memory

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!*)

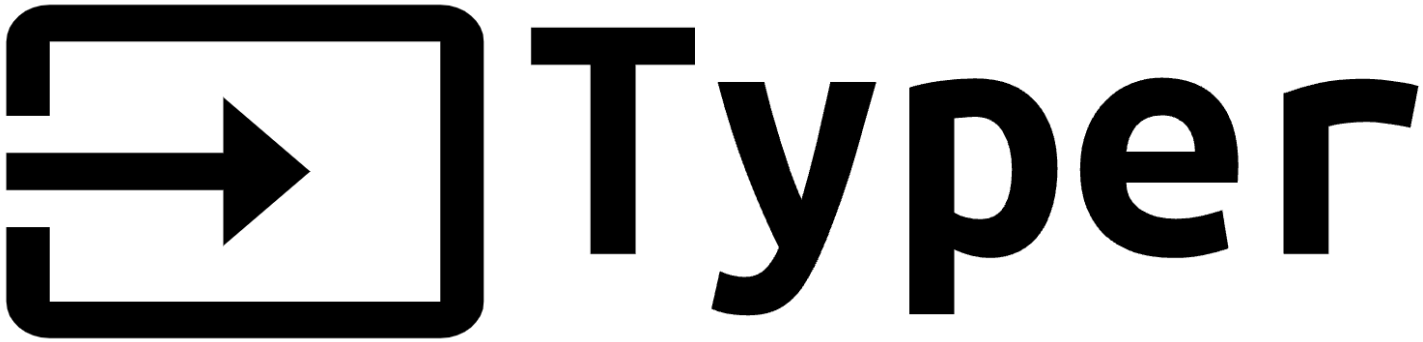
Discrete Structures with Python

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 language resembles the mathematics that you might employ in your work!)

Python Resource

Typer



`https://typer.tiangolo.com/`

Typer

- 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/`

Poetry

- 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

Python Resource

JupyterLite

`https://jupyter.cs.allegheny.edu/hub/login`

JupyterLite

- Online Python version 3 on Departmental Server
- Your own machine in the cloud with a command prompt for `git`
- To log-in, you must have a GitHub account.
- Dept of CS videos to help: `https://www.youtube.com/playlist?list=PLsYZRXov75ZHSwWiCk0-jd1RcTuu_-zmD`

About Python

- `www.python.org`
- Download python3 if you are using your own hardware

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.

Key Components

All programs built out of ...

- **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
```

Programming Constructs

Key Questions

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?

Learning Objectives

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

A program is a sequence of statements

To be philosophical for a moment ...

Going back to this program ...

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

Programming parallels cooking ...

- 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

$$ax^2 + bx + c = 0 \tag{1}$$

Quadratic Formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

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 Root Calculation

The Problem Defined

To solve

$$x^2 + 3x - 4 = 0 \tag{3}$$

Want to have roots

$$x_1 =? \text{ and } x_2 =?$$

Finding Roots of an Equation

Let's work an example by hand ...

Quadratic Formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{4}$$

Two solutions: x_1 and x_2

Knowns: a = 1, b = 3, c = -4

$$x_{1,2} = \frac{-3 \pm \sqrt{(-3)^2 - 4(1)(-4)}}{2(1)} \tag{5}$$

$$x_{1,2} = \frac{-3 \pm \sqrt{25}}{2} \tag{6}$$

1. **Root 1** using addition of $\pm$: $x_1 = \frac{2}{2} = 1$,
2. **Root 2** using subtraction of $\pm$: $x_2 = \frac{-8}{2} = -4$

Functions

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?

Learning Objectives

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

Absolute Value of a Number

A function to calculate value

Function for finding absolute values

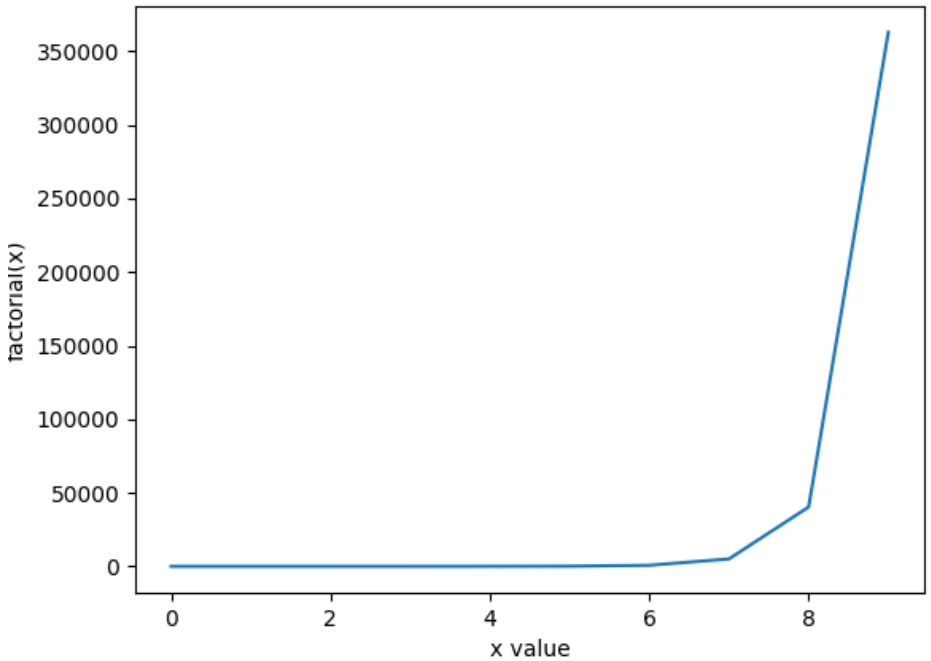
```
def abs(n):
    if n >= 0:
        return n
    else:
        return -n
```

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

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

Factorials: another definition

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

- Factorials are applied to integers

Factorials

Factorials

$N! = N * (N-1) * (N-2) * ... * (2) * (1)$

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

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

$3! = 3 * 2 * 1$

$2! = 2 * 1$

$1! = 1$

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

Factorials defined

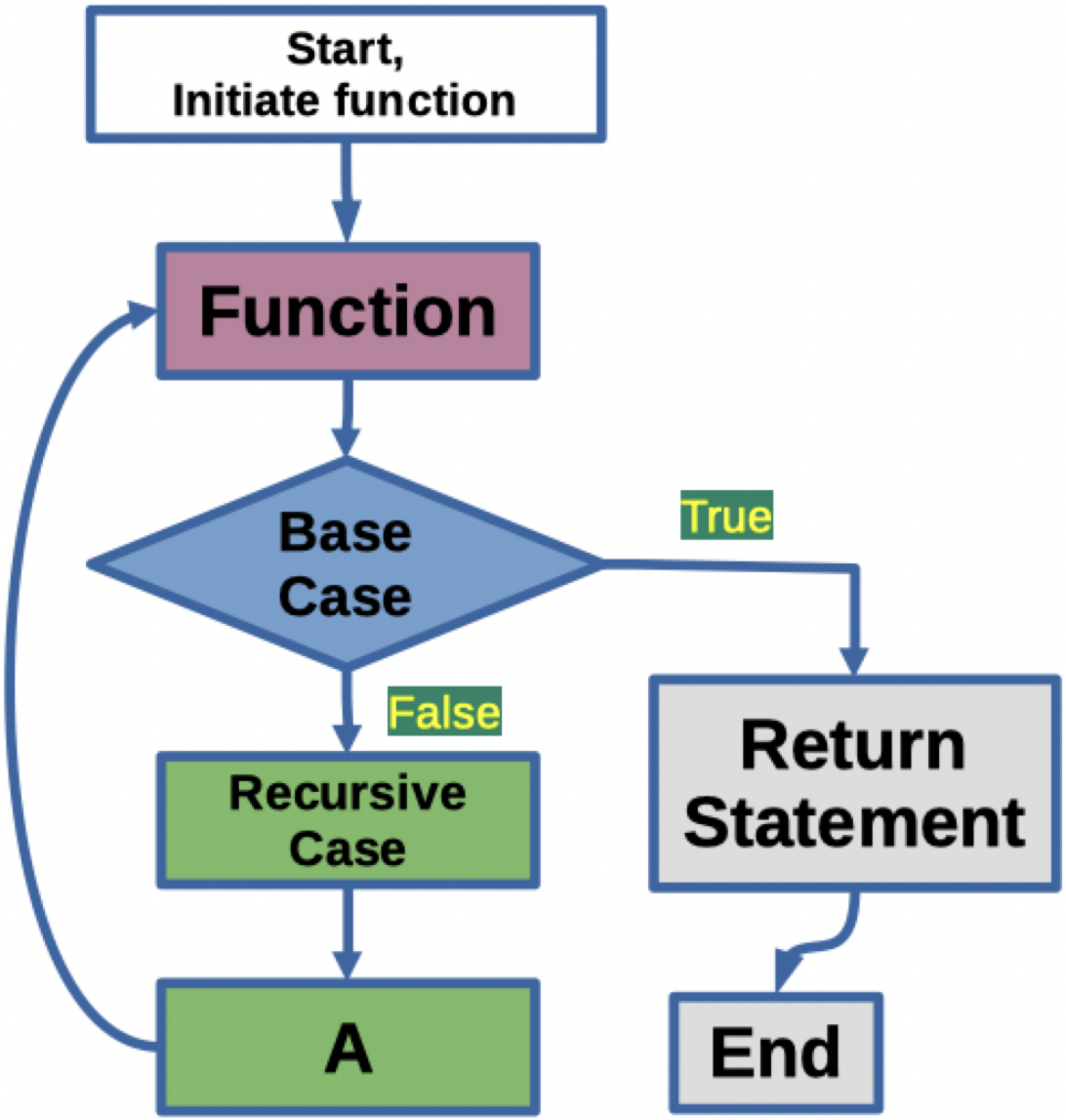
$N! = [\ (N-1)! + (N-2)!\ ] * (N-1)$

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

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

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

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?

The Code

```
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 does this function make progress to the base case?

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))
```

- 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

Virtual Environments

Key Questions

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

Learning Objectives

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/`

Setup Steps

Add File: projects/hello_user/hello_user/__init__.py

```
"""Required docstring for an __init__ file."""

__version__ = "0.1.0"
```

Add File:projects/hello_user/pyproject.toml

```
[tool.poetry] ...

[tool.poetry.scripts]
hello_user = "hello_user.main:cli"

[tool.poetry.dependencies] ...
```

Update Poetry

```
poetry install
```

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"   first = {first}")
    console.print(f"   middle = {middle}")
    console.print(f"   last = {last}")
# end of main()
```

Basic Reformatting with Black

```
poetry run black hello_user tests
```

```
obonhamcarter@MacBookPro-2017 hello_user % poetry run black hello_user tests
reformatted hello_user/main.py

All done! ✨ 🍰 ✨
1 file reformatted, 2 files left unchanged.
```

Execute Project

What do you see?

```
# run from projects/hello_user
poetry run hello_user --help
```

```
obonhamcarter@MacBookPro-2017 hello_user % poetry run hello_user --help

 Usage: hello_user [OPTIONS]

 Say hello to the person having a name of first, middle and last name

╭─ Options ──────────────────────────────────────────────────────────────────╮
 --first                      TEXT
 --middle                     TEXT
 --last                       TEXT
 --install-completion         [bash|zsh|fish|powershell  Install completion for
                              |pwsh]                    the specified shell.
                                                        [default: None]
 --show-completion            [bash|zsh|fish|powershell  Show completion for the
                              |pwsh]                    specified shell, to copy
                                                        it or customize the
                                                        installation.
                                                        [default: None]
 --help                                                 Show this message and
                                                        exit.
╰────────────────────────────────────────────────────────────────────────────╯
```

Ordered Pairs

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?

Learning Objectives

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

What are *Ordered Pairs*?

Some definitions

- 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?

Moniods

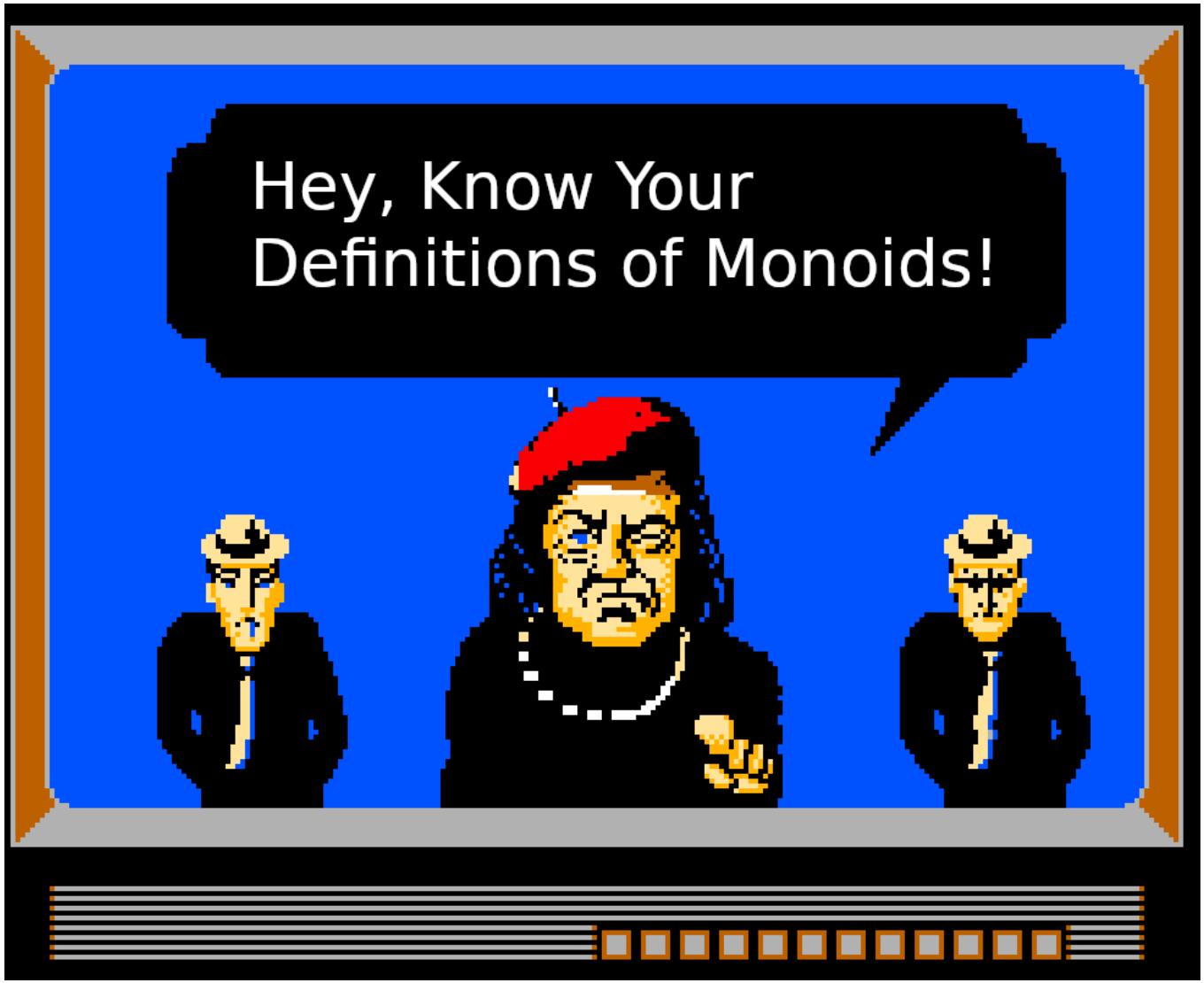
Key Questions

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?

Learning Objectives

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

And Now This TV



A Quick Definition

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`

Finite Sets

Key Questions

How do I use **dynamically generated streams** of data to implement **memory efficient** and **predictable** Python programs?

Learning Objectives

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

Project to Make

Project Guide Lines

- Want to make am infinite palindrome sequence generator.

Format

- Format: ABCBA

Examples of palindromes

- 11,22, 33, ...
- 2824282, 2825282, 2826282, 2827282, 2828282, ...
- 478874, 479974, 480084, 481184, 482284, 483384, ...,
- 6513156, 6514156, 6515156, 6516156, 6517156, ...

Add File: projects/makepal/makepal/main.py

Use file located in sandbox: `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()
```

Add File: projects/makepal/makepal/main.py

Use file located in sandbox: `main.py`

```
@cli.command()
def main(upperbounds: str = ""):
    """Driver function. Upperbounds
    is how high we go to create palendromes"""

    for i in infinite_sequence():
        pal = is_palindrome(i)
        if pal:
            # print(f"\t {count}, {pal}")
            print(f"{pal}")

# end of main()
```

Add File: projects/makepal/makepal/main.py

Use file located in sandbox: `main.py`

```
def is_palindrome(num):
    # Skip single-digit inputs
    if num // 10 == 0:  # return an int, not a float
        return False
    temp = num
    reversed_num = 0

    while temp != 0:
        reversed_num = (reversed_num * 10) + (temp % 10)
        temp = temp // 10

    if num == reversed_num:
        return num
    else:
        return False

# end of is_palindrome()
```

Add File: projects/makepal/makepal/main.py

Use file located in sandbox: `main.py`

```
def infinite_sequence() -> None:
    """Infinite_sequence
    will eventually stop at an upperbounds"""

    num = 0
    count = 0
    while True:
        yield num
        num += 1

# end of infinite_sequence()
```

YIELD instead of RETURN

Sets

Key Questions

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

Learning Objectives

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

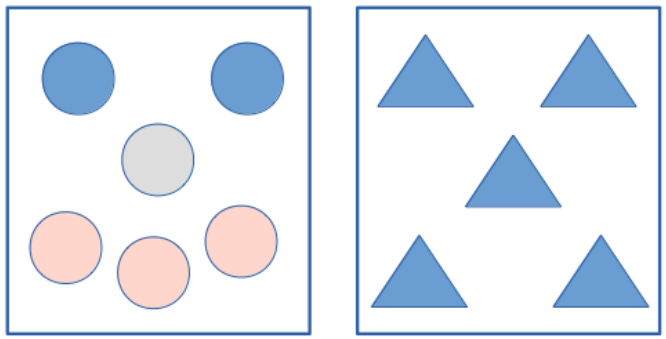
Intensional Sets

A list of characters in Sherlock Holmes

- {Sherlock Holmes, Dr. John Watson, D.I. Greg Lestrade, Mrs. Hudson, Mycroft Holmes, Irene Adler, Mary (Morstan) Watson}

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)
- $F = \{n^2 - 4 : n \text{ is an integer; and } 0 \leq n \leq 19\}$
  - The set of all values gained from plugging in n between 0 and 19 into the equation $n^2 - 4$

Probabilities

Key Questions

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

Learning Objectives

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

Creating a Set from a List or Tuple

```
import sympy as sy

list = [2, 4, 6, 8, 10]
finite_set = sy.FiniteSet(*list)
print(finite_set)

tuple = (2, 4, 6, 8, 10)
finite_set = sy.FiniteSet(*tuple)
print(finite_set)
```

- All approaches call the **FiniteSet** constructor
- Can construct a **FiniteSet** out of a list or a tuple
- What is the purpose of the ‘*‘ in this program?

Understanding Outputs

Output of Finite Set Creation Program

```
import sympy as sy

# Explicit FiniteSet:
sy.FiniteSet(2, 4, 6, 8, 10)

# Empty FiniteSet:
EmptySet

# FiniteSet from Tuple:
sy.FiniteSet(2, 4, 6, 8, 10)

# FiniteSet Containing Tuple:
sy.FiniteSet((2, 4, 6, 8, 10))
```

Creating Sets

Using Finite Sets in Sympy

```
from sympy import FiniteSet

list = [1, 2, 3, 2]
finite_set = FiniteSet(*list)
print(finite_set)

for element in finite_set:
    print(element)
```

- What is the output of `print(finite_set)` ?
- What is the output of `print(element)` in the `for` loop?
- How do these two output segments differ?

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 both prime **or** odd? To determine this, you calculate the probability of the **union** of the two event sets.
$E = A \bigcap B \{2, 3, 5\} \bigcap \{1, 3, 5\} = \{1, 2, 3, 5\}$

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.union(roll_two)
prob = len(event) / len(six_sided)
print(prob)
```

- The 'intersect' function connects to a logical 'and' operation
- The output of this program is `0.6666666666666666`. Why?
- Could also make a direct call to the 'probability' function!

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.
$E = A \bigcup B \{2, 3, 5\} \bigcup \{1, 3, 5\} = \{1, 3, 5\}$

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)
print(prob)
```

- The 'intersect' function connects to a logical 'and' operation
- The output of this program is `0.3333333333333333`. Why?
- Could also make a direct call to the 'probability' function!

AND SO MUCH MORE!!