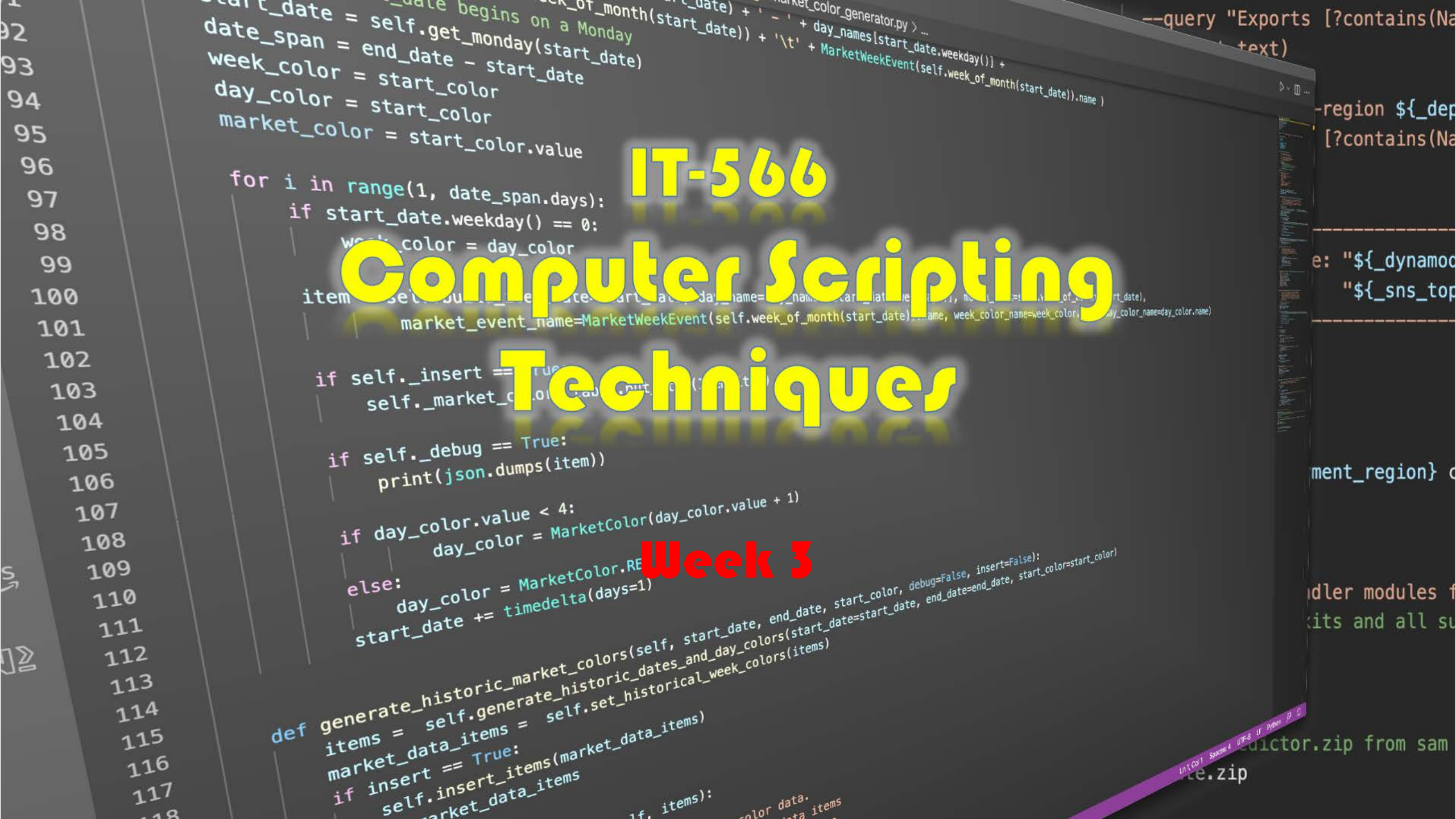

IT-566
Computer Scripting
Techniques
Week 3

Week 3

Talking Points

- Problem Solving
- How To Learn Python or Any Programming Language
- Fundamentals
- Processing Command-Line Arguments


What Do Computers Do Best?

Answer

Repeated calculations...

What Differentiates Humans From Computers?

Answer

Abstract thought...




What's the hardest part about programming a computer?

Answer

- Analyzing a problem
- Formulating a solution
- Bounding solution with constraints
- Implementing solution to run on a machine

Problem Solving

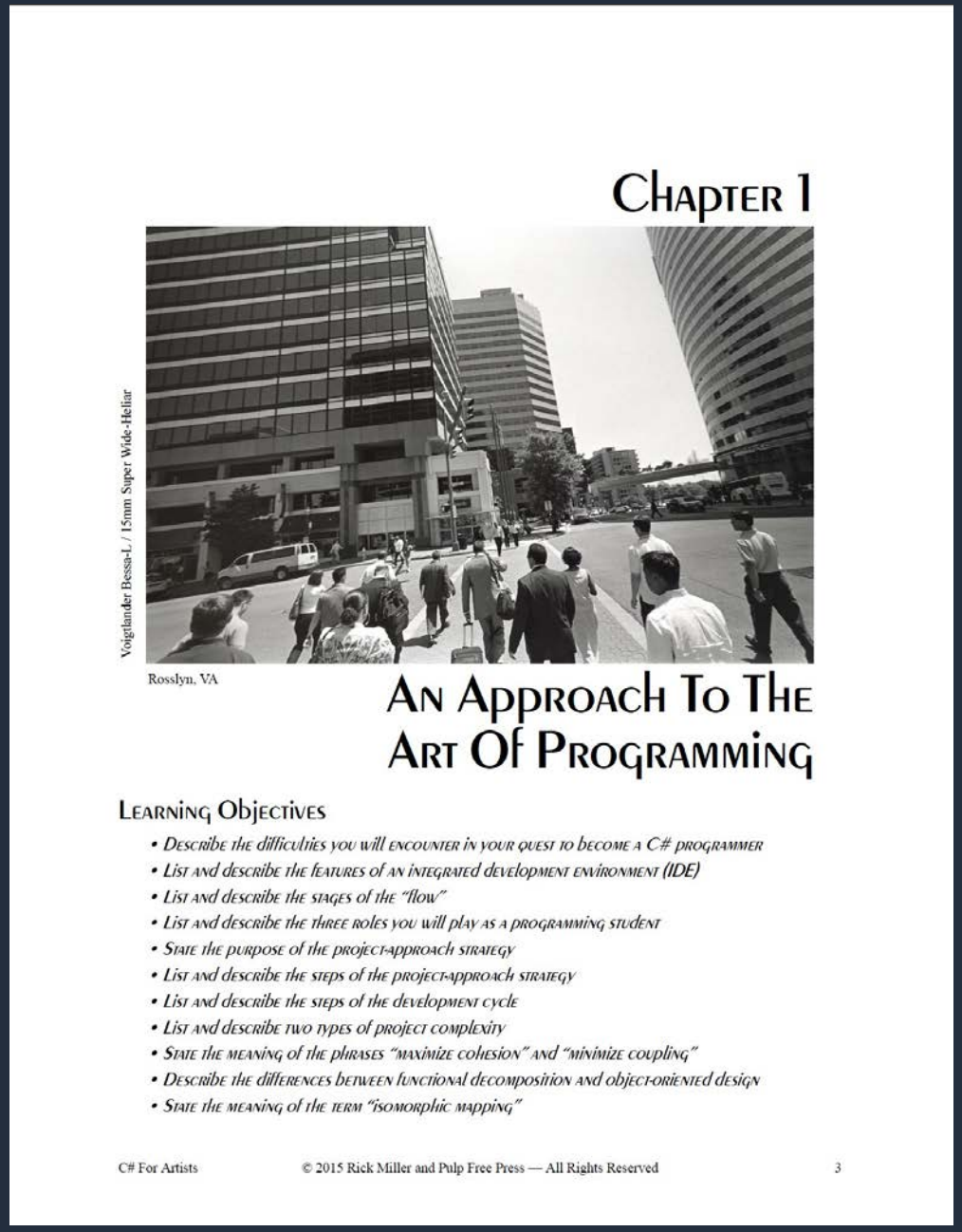
A General Approach

Material from C# For Artists, Chapter 1

Difficulties You'll Encounter…

In Addition to the Syntax and Semantics of Python…

- A development environment, which could be as simple as the combination of a text editor and command-line compiler or as complex as a commercial product that integrates editing, compiling, and project management capabilities into one suite of tools
- A computing platform of your choice (i.e., a compute running Microsoft Windows 10, MacOS, or Linux.)
- Problem solving skills
- Project approach techniques
- Project complexity management techniques
- The ability to put yourself in the mood to program
- The ability to stimulate your creativity
- Object-oriented analysis and design
- Object-oriented programming principles
- Functional Decomposition



CHAPTER 1

Voigtlander Bessa-L / 15mm Super Wide-Heliar

Rosslyn, VA

AN APPROACH TO THE ART OF PROGRAMMING

LEARNING OBJECTIVES

- *Describe the difficulties you will encounter in your quest to become a C# programmer*
- *List and describe the features of an integrated development environment (IDE)*
- *List and describe the stages of the "flow"*
- *List and describe the three roles you will play as a programming student*
- *State the purpose of the project-approach strategy*
- *List and describe the steps of the project-approach strategy*
- *List and describe the steps of the development cycle*
- *List and describe two types of project complexity*
- *State the meaning of the phrases "maximize cohesion" and "minimize coupling"*
- *Describe the differences between functional decomposition and object-oriented design*
- *State the meaning of the term "isomorphic mapping"*

Personality Traits of Great Programmers

Creative	The most prevalent trait. Solving problems such that they run on a computer requires lots of creativity.
Tenacious	Do you like to bite into a problem like a pit bull and not let go until you succeed?
Resilient	When a though problem gives you a thorough trouncing, do you take a break and come back strong?
Methodical	You must approach a solution in a methodical way, whether a formal methodology exists or not.
Meticulous	Close attention to detail is paramount. You will suffer at first until you learn the lesson.
Honest	Do the right thing in the code when no one is looking.
Proactive	Recognize and capitalize on opportunity. Ask questions. Seek answers.
Humble	Know when to seek guidance or help. Push ego aside for the greater good.

Be A Generalist and a Just-In-Time Specialist

"Great programmers are well-versed in all aspects of computing. Rarely have I ever met any who referred to themselves as only a this type of programmer or a that type of programmer. I'd rather hire generalists with solid educational backgrounds and the proven ability to teach themselves new tricks, than to bank on a specialist who refuses to grow professionally. In other words, great programmers have a broad range of skills they can apply to the problem. Great programmers can gather requirements, design a solution, write the code, conduct testing, write supporting documentation, deploy the application, if necessary, and carry on intelligent conversations with the customer to boot."

Three Roles You Play

Analyst	You need to formulate a deep understanding of the problem you are trying to solve. Don't understand a programming assignment? Ask questions. Study the problem. Become a subject matter expert (SME).
Architect	Design a solution. Must understand software design techniques. A simple project may rest easily upon a simple design, but if multiple objects interact with each other or clients must communicate with servers, or millions of requests must be processed simultaneously, or huge amounts of data must be ingested and processed within seconds, such systems will demand completely different architectures.
Programmer	You must implement your design. Soon enough you will discover if you've designed yourself into a corner. You may need to refactor your code and your design.

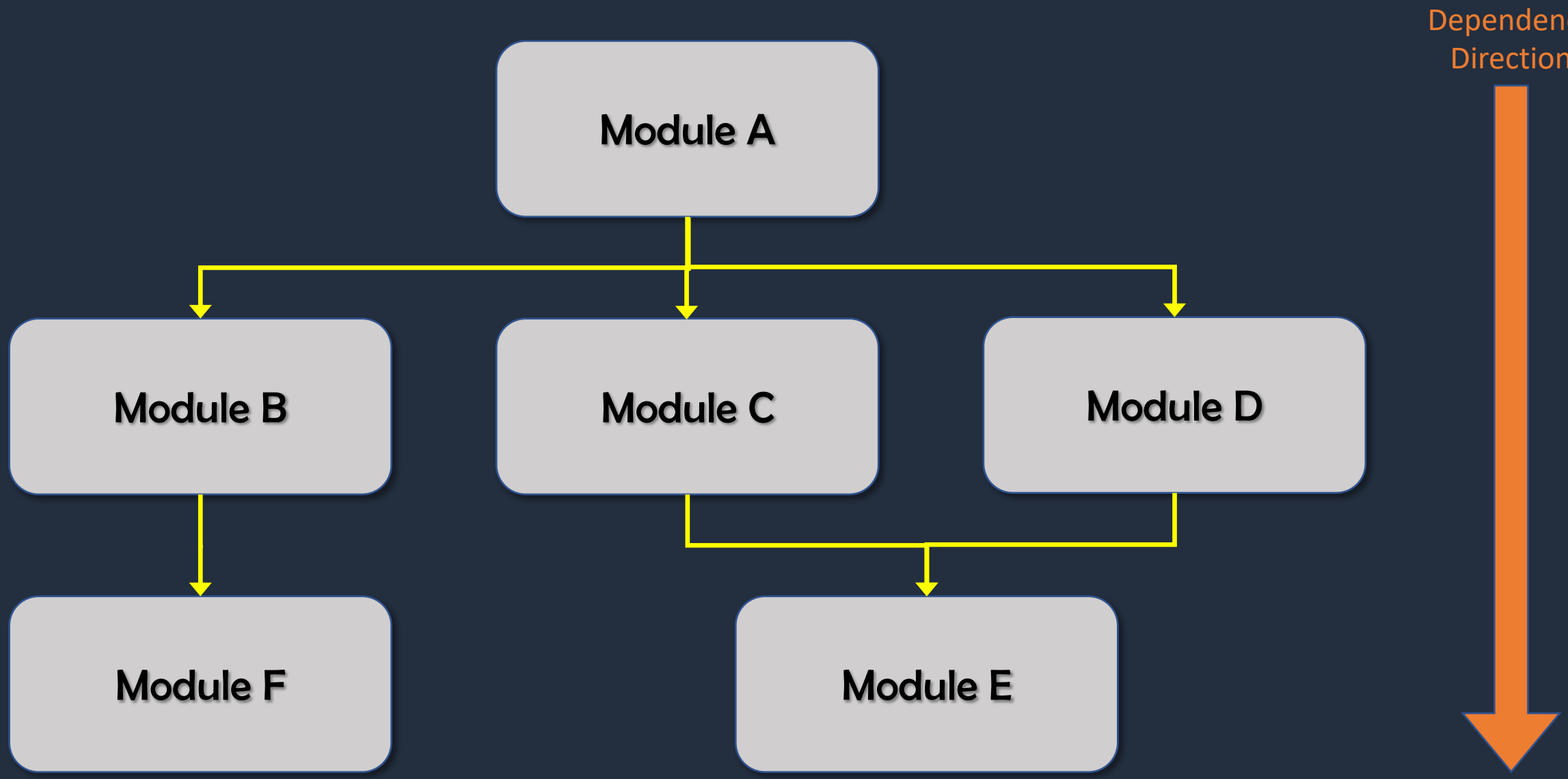
Project Approach Strategy

- Helps You To…
  - Get Started
  - Maintain Forward Momentum
  - Overcome Obstacles
  - Get Unstuck
  - Maintain Positive Attitude
  - Avoid Depression
  - Grow Professionally
  - Feel Good About Your Chosen Career Path
  - Enjoy Programming
  - Conquer The World
  - Become Ruler of the Universe

Project Approach Strategy — Areas of Concern

Application Requirements	An assertion about expected behavior. Contained in project specification or programming assignment. (*Real world requirements derived from many possible sources.*) Ensure complete understanding before proceeding. Seek clarification. Ask questions.
Problem Domain	Body of knowledge required to implement a software solution apart and distinct from the knowledge of programming itself. ”Write an elevator simulation program…” You may know nothing about elevators. You need to become enough of an expert to understand the issues involved. Also known as Subject Matter Experts (SMEs)
Programming Language Features	Major source of frustration, especially in the beginning. To save yourself from panic, make a list of the language features you need to understand, marking it off the list as you go. Provides focus and a sense of progress. As you study each feature, take notes on its usage. Refer to your notes when you begin to formulate your design.
High-Level Design	A.K.A. — Application Architecture. Might be a single file. Might be hundreds. Various approaches: Procedural, Functional, Object-Oriented.
Implementation Strategy	How you intend to code the program? Where will you start? How much code should your write at a time? Will you write everything from scratch or stand on the shoulders of giants? When you start writing code you need to stick to a **development cycle**. Be methodical.

Functional Decomposition & Procedural Programming



Functional Decomposition & Procedural Programming

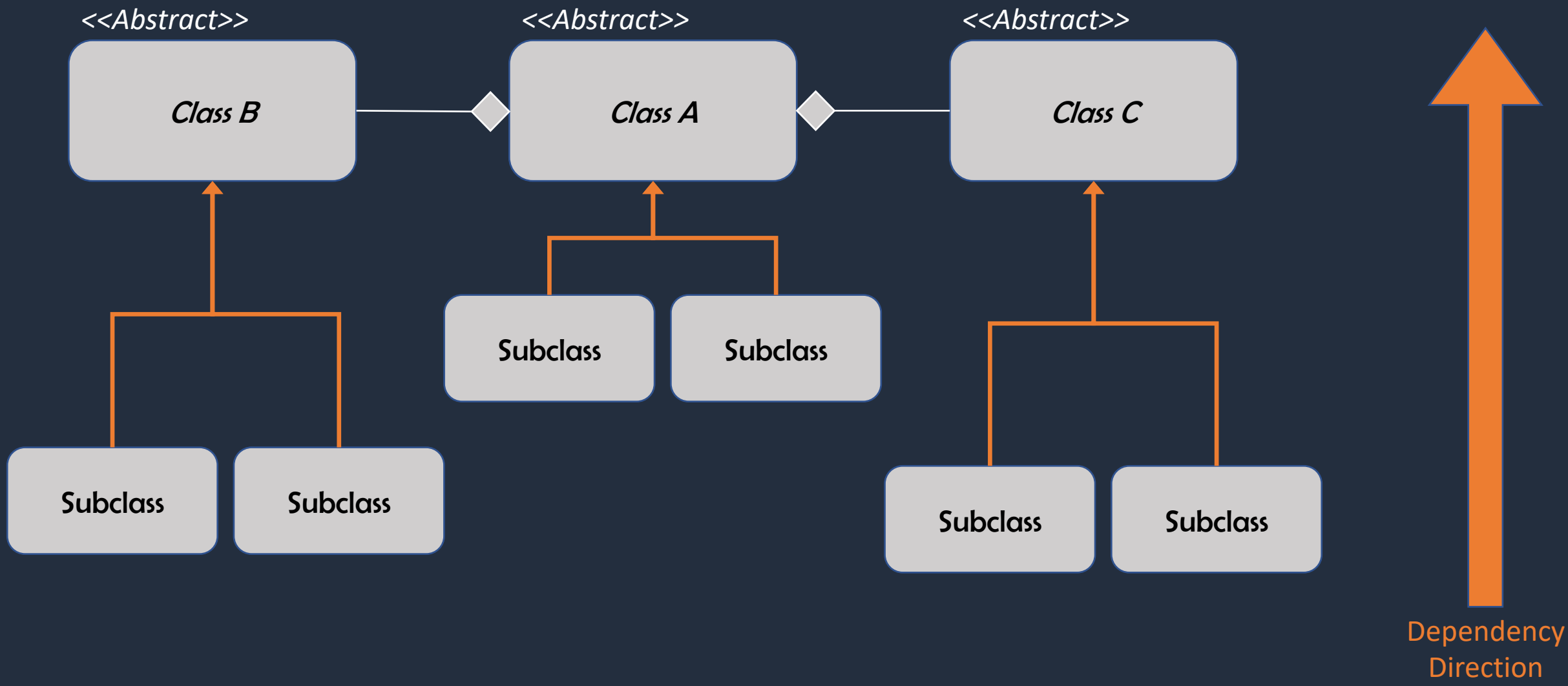
Pros

- Naturally follows from traditional business analysis techniques
- Process and feature requirements map to modules and functions
- Quick way to implement features
- Easy to understand – Notionally

Cons

- Top-down dependencies
  - Module A depends on submodules
- A change to a submodule…
  - …affects calling module
- Difficult if not impossible to isolate modules for reuse
- Hidden or unknown dependencies may break code in unexpected ways
- New features require modifying existing code

Object-Oriented



Object-Oriented Analysis, Design, and Programming

Pros

- Model interaction between real world objects
  - Isomorphic Mapping
- Stable, well-defined abstractions
- Extend class vs. modify code to add functionality
  - Open Closed Principle (OCP)
- Inverted dependency hierarchy
  - Dependency Inversion Principle (DIP)

Cons

- Steep learning curve
- Takes practice to get high-level abstractions right
- Programs tend to have more boilerplate code
- Language needs to support accepted OOP features
  - Encapsulation
  - Virtual Calling
  - …

Development Cycle

Plan	Design to the point where you can start coding. Do not attempt to design everything up front. The idea here is to keep your design flexible and open to change.
Code	Implement what you have designed.
Test	Thoroughly test each section or module of source code. The idea here is to try to break it before it has a chance to break your application. Write unit tests. Aim for >85% code coverage. When you write a piece of code ask yourself, “How can I test this?” If testing seems difficult or impossible, perhaps you need to refactor.
Integration & Regression Testing	As you iterate over each development sprint perform integration and regression testing. This simply means as you create individual pieces of functionality in your application, start testing them together where possible.
Refactor	When you encounter difficulties with your design and implementation, or you think of a better way of doing something, make necessary improvements.

Code Stages

Stage	A.K.A.	Description
Just Get It Working	Ugly Baby Gnarly Toenail	Works. Proof of concept. Full of code smells. Not pretty. Not optimal. Influenced by bad habits. Untested. Exploitable. Bad formatting. No comments. Magic values. Hard to understand. Impossible to change without breaking something.
Transitional	Conformant	Starting to look like Python. Best practices. Self commenting. Consistent naming convention.
Idiomatic	Beautiful Code	Well architected. Well structured. More efficient. Minimally coupled, Maximally cohesive. Easy to comprehend. Easy to modify.
Optimized	Quality Without A Name (QWAN)	Every statement analyzed for speed and resource usage. Optimized for maximum efficiency.

How To Learn Python

or

Any Programming Language

Programming In Python Isn't Hard

Programming In Python Is Hard

Hard isn't really the right word, but...

You'll spend way more time and effort learning to program vs. learning Python.

If you're new to programming, you're learning two new concepts: 1. How to program, and 2. Python. That's hard.

Once you learn how to program, learning another programming language is easy.

But there are caveats...

Depends on 1. Your first programming language, and 2. The paradigm it supported, and you adopted, to design and implement your programs.

By paradigm I mean:

- Imperitive
  - Machine Code
  - Assembly
  - Procedural
  - Object-Oriented
- Declarative
  - Functional

What's The Difference?

- Imperitive
  - Step-by-Step
  - Control Flow (Explicit)
- Declarative
  - Data Flow
  - Desired Results

Python Supports Multiple Paradigms to Certain Degrees

Imperative Procedural

```
import boto3
import json
import os
import re

def lambda_handler(event, context):
    print("******* Begin Execution *********")
    print("EVENT: " + json.dumps(event))
    json_region = os.environ['AWS_REGION']
    print('JSON Region: ' + json_region)
    message = {}
    ip = '0.0.0.0'
    userAgent = event['requestContext']['identity']['userAgent']
    print('USER-AGENT: ' + userAgent)


    # If Amazon-Route53-Health-Check-Service skip sqs and sns
    if "Amazon-Route53-Health-Check-Service" not in userAgent:
        # Do something here...


        try:
            # Extract user's IP IpAddress from X-Forwarded-For header
            try:
                results = re.findall("^\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}", event['headers']['X-Forwarded-For'] )
                print("USER'S IP ADDRESS: " + results[0])
                ip = results[0]
            except Exception as e:
                print("ERROR EXTRACTING IP ADDRESS: " + str(e))

            # Extract queryStringParameters
            try:
                if 'queryStringParameters' in event:
                    if event['queryStringParameters'] is not None:
                        if 'r' in event['queryStringParameters']:
                            if event['queryStringParameters']['r'] is not None:
                                message['range'] = event['queryStringParameters']['r']
                            else:
                                message['range'] = '1'
                        if 'u' in event['queryStringParameters']:
                            if event['queryStringParameters']['u'] is not None:
                                message['unit'] = event['queryStringParameters']['u']
                            else:
                                message['unit'] = 'd'
```

Imperative Object-Oriented

```
from abc import ABC, abstractmethod
from powerplants.powerplant import PowerPlant
from weapons.weapon import Weapon


class Vessel(ABC):
    """
    Defines the interface for a Weapon object.
    """
    def __init__(self, identifier: str, weapon: Weapon, powerplant: PowerPlant):
        self._identifier = identifier
        self._weapon = weapon
        self._powerplant = powerplant
        print("%s %s: base class object created" % (self.__class__.__bases__[0].__name__, self._identifier))

    @abstractmethod
    def aim(self):
        print("%s: aim() method called..." % (self.__class__.__bases__[0].__name__))

    @abstractmethod
    def fire(self):
        pass

    @abstractmethod
    def lightoff_plant(self):
        pass

    @abstractmethod
    def shutdown_plant(self):
        pass

    @abstractmethod
    def __repr__(self):
        return "%s" % (self.__class__.__name__ + " " + self._identifier)
```

Declarative Functional

Iterators

```
def iter_demo(self):
    message = 'A string of letters is also a list of chars...'
    for s in message:
        print(f'{s} ', end="")
```

Lambda, map()

```
def lambda_demo(self):
    print(f'{list(map(lambda x: x + x , [1,2,3,4,5] ))} ', end="")
```

Dynamic Typing
(No Static Typing)

No Encapsulation

Not Purely Functional, but
Functions are First Class Objects

To Learn Python — Learn [*How To…*]

- Find Answers
  - Python Documents
  - Google
  - YouTube
- Organize Your Code
  - Packages
  - Modules
  - Classes
  - Functions
- Create Code
  - IDE
- Format Code
  - PEP 8
- Run Programs
- Debug Code

- Test Code
- Name Things
- Store data in:
  - Variables
  - Constants
- Scoping Rules
- Operators
- Loop (Iterate)
  - for
  - while
- Branch (Conditions)
  - if/else
  - match/case

- Data Structures
  - Tuples
  - Lists
  - Dictionaries
  - Sets
- Built-In Functions
- Data Input/Output (I/O)
- File Processing
- Handle Exceptions
- Use 3rd Party Libraries
- Break Old Habits
  - Python Idioms

Pro Tips

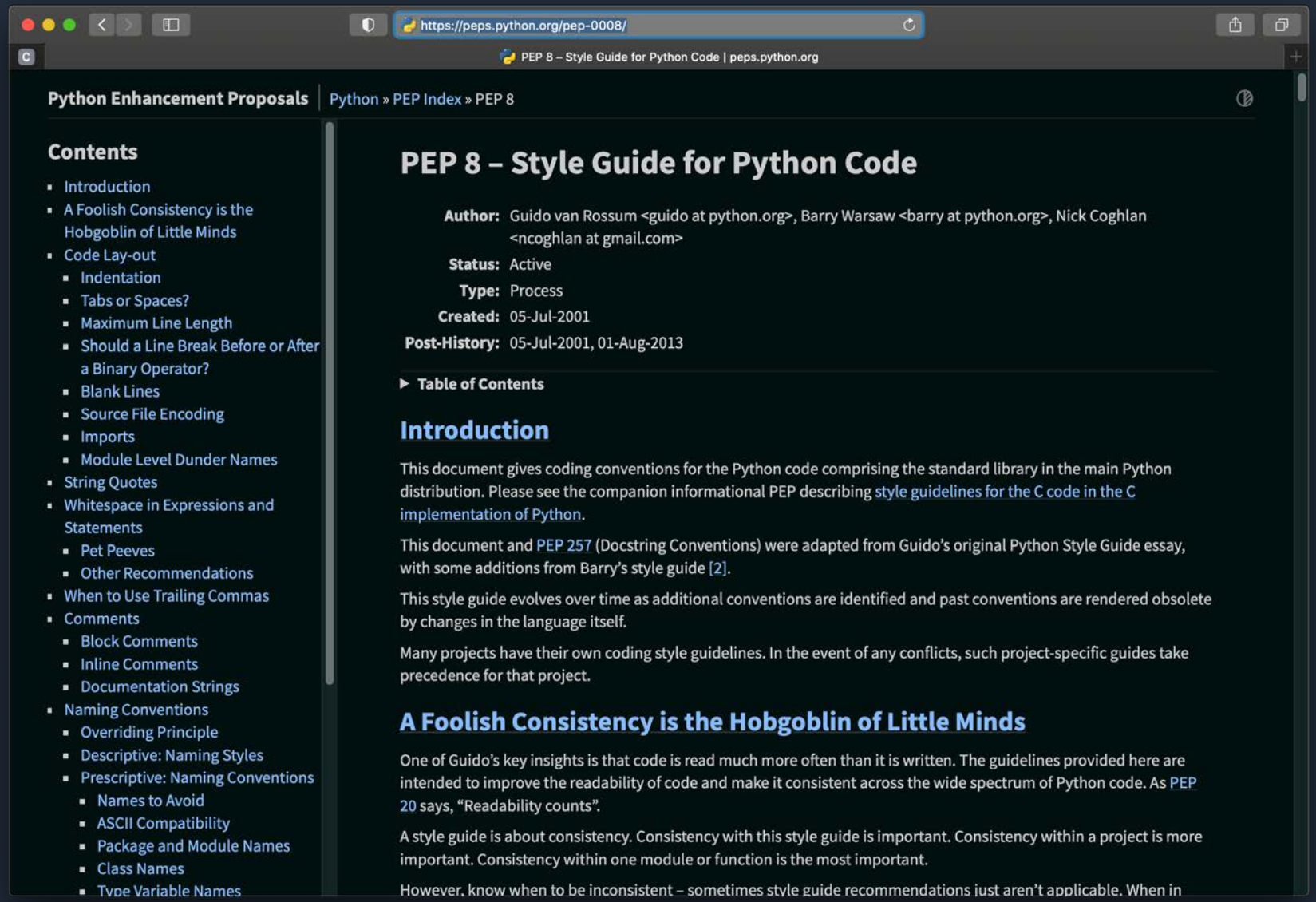
- Don't Start At the Computer
  - Ever write an essay?
  - Before you sit down to write code…
    - You should have a good idea how to proceed
- Think Algorithm First
  - Write comments in plain English
    - Pseudo Code
  - Becomes intuitive over time
- Translate comments into code
- Run Early and Often
  - Fix the first error first
  - Then try again

But Most Importantly…

When Searching for Help — Ask Google Straight Up What You're Trying To Do In Clearest Most Direct Way Possible

Fundamentals

PEP-8 — Style Guide for Python Code

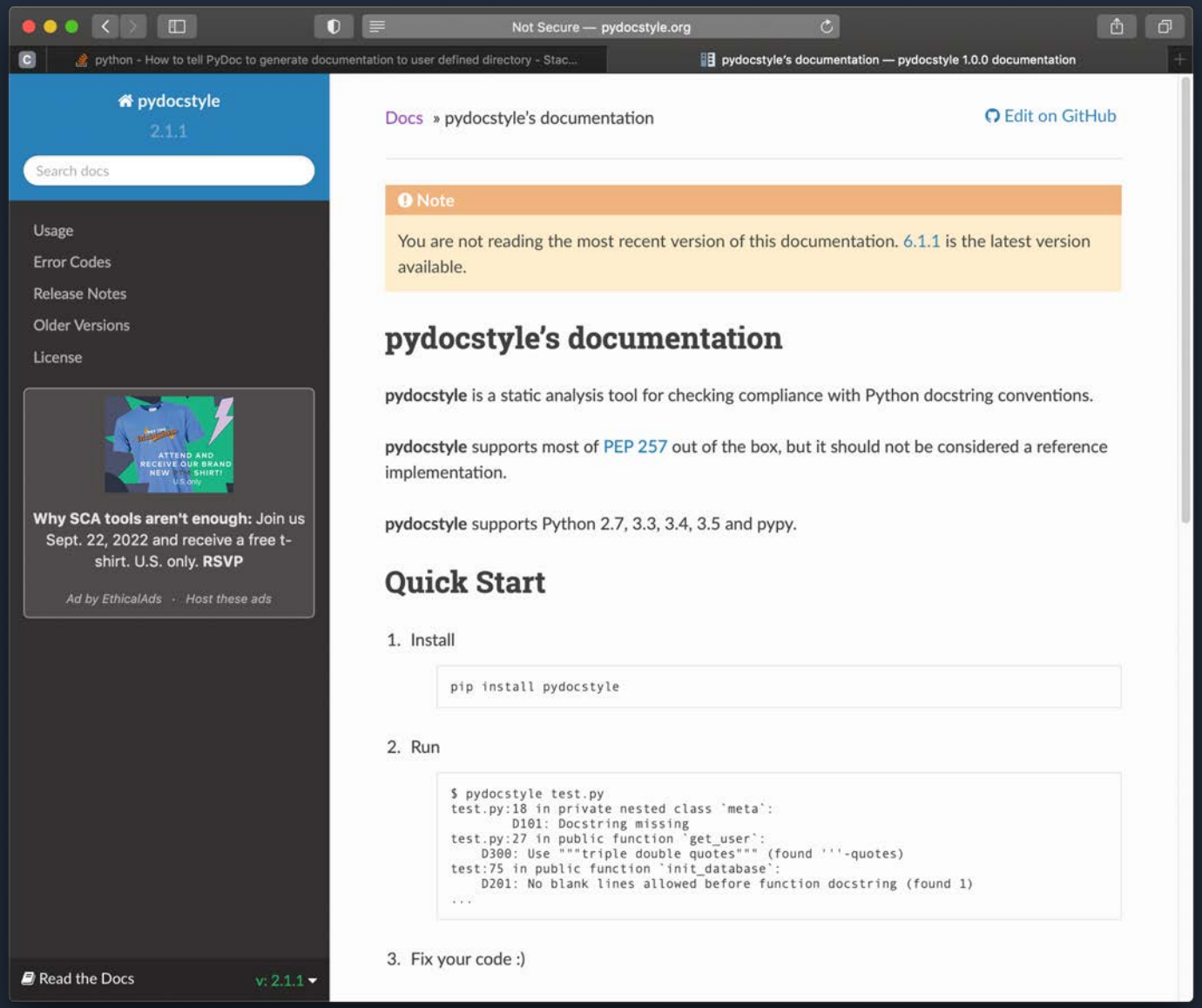


https://peps.python.org/pep-0008/

Focus On…

- Code Layout
- Consistent String Quoting
- Naming Conventions
  - Packages
  - Modules
  - Functions
  - Classes
  - Methods
  - Variables
  - Constants
- Plugins Can Help with Formatting

Doc Comments



pydocstyle

http://www.pydocstyle.org

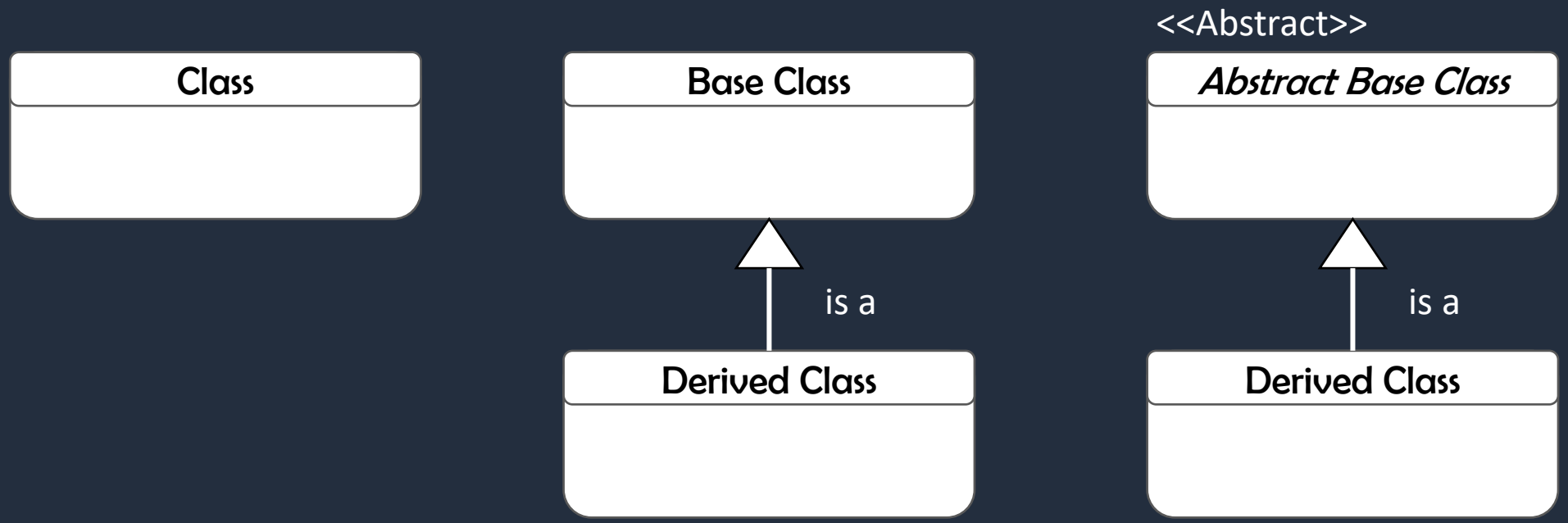
- Validates doc comments
- Installation:
  pipenv install –dev pydocstyle
- Run
  pipenv run pydocstyle src/example.py

Classes & Methods

- Classes
  - Logical Organization and Way of Thinking
    - Object-Oriented Analysis, Design, and Programming (OOAD & P)
  - Physical Namespace for Methods and Attributes
  - Group Together Related Code and Data
- Methods
  - Functions Associated with a Class
    - *A function defined outside of a class is called a function*
    - *A function defined inside of a class is called a method*
- A Program is an Interaction Between Objects
- Rule of Thumb
  - One Class Per Module — For Sanity
  - Module and Class Have Same Name
    - Module name all lowercase | Class name begins with upper case letter

OOAD&P Design Language — UML

UML — Unified Modeling Language



UML is more than just pictures — I mostly use it to communicate designs

Sequence Types: Lists, Tuples, Ranges

https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

List [ ]

- An iterable, mutable sequence of objects
- List is an object with methods

  `[1,2,3,4].append(5) => [1,2,3,4,5]`
- Some operations on lists performed via global built-in functions

  len([1,2,3,4,5]) => 5

  min([1,2,3,4,5]) => 1
- Accessed via index

  [1,2,3,4,5][0] => 1
- Strings can be treated like lists but are immutable
  - Don’t support operations that would change the state

  ’Hello, World!’[0] => H

Dictionaries

https://docs.python.org/3/tutorial/datastructures.html#dictionaries

- Dictionary { }
  - Iterable set of *key*:*value* pairs
  - Keys must be an immutable type
    - string
    - integer
    - tuple (As long as it contains immutable types)
  - Operations

```
ages = { }                          # Create empty dictionary
ages = {'Bill':23, 'Steve':42, ...} # Create with data
ages['Bill']                        # Access via key => 23
list(ages)                          # Returns list of keys
ages['Coralie'] = 32                # Add new key:value
```

File Input/Output (I/O)

https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

- Easy-Peasy in Python

```
f = open('names.txt', 'w', encoding='utf-8')
f.write('Guido van Rossum')
f.close()
```

- But Use with Keyword (Automatic Resource Cleanup)

```
with open('names.txt', 'r', encoding='utf-8') as f:
    names_data = f.read()
```

Exceptions

https://docs.python.org/3/tutorial/errors.html

- Lots Of Things Can Go Wrong In Code
  - File I/O Errors
  - Network Errors
  - Just to list a few…
- Exception
  - Error Detected During Execution
- `try/except` Statement
  - Place Code in Try Block
  - Handle Exceptions in Except Block

https://docs.python.org/3/library/exceptions.html#concrete-exceptions

Processing Command-Line Arguments

with

`argparse`

Backup Slides
