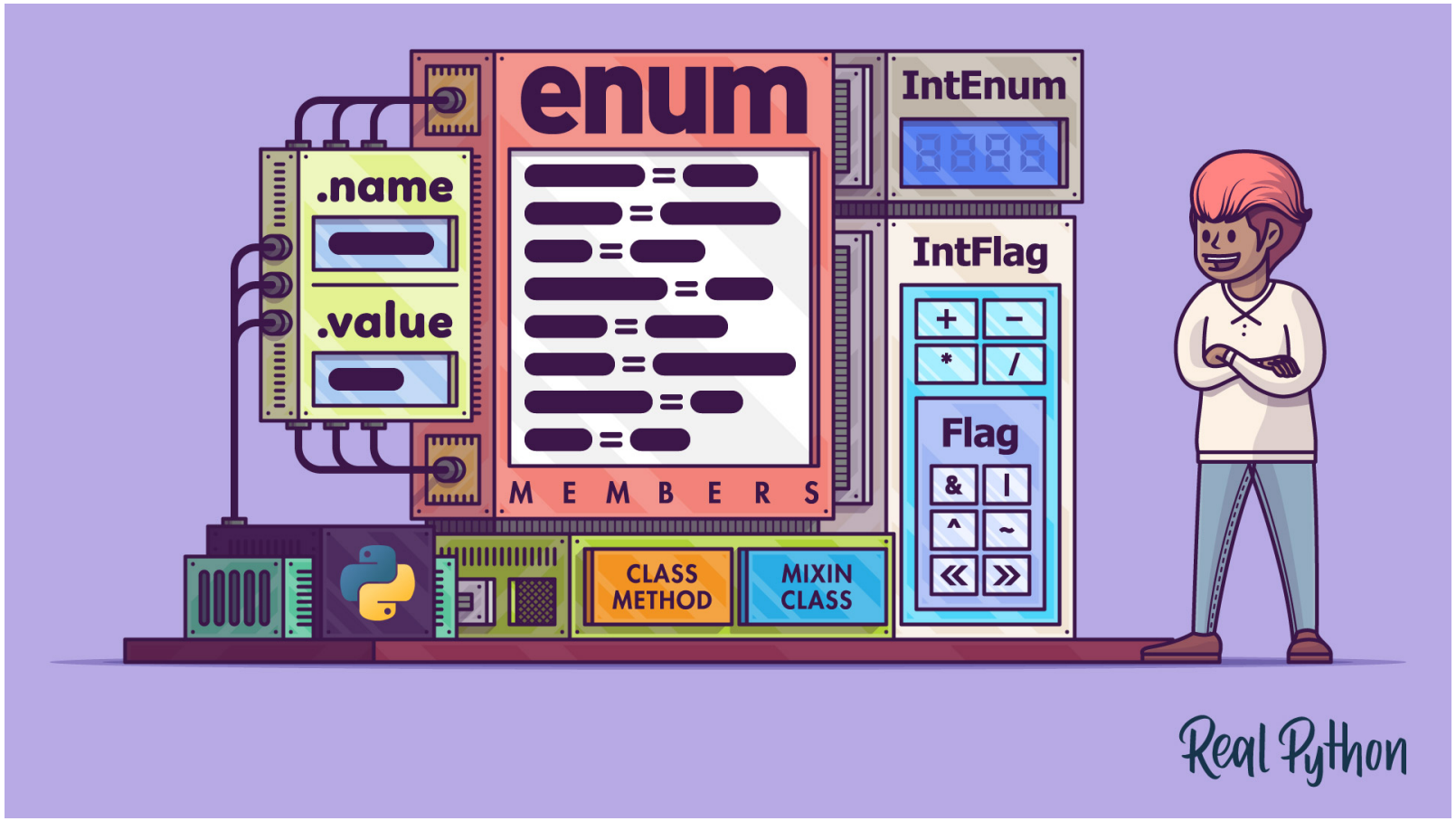
Building Enumerations With Python's `enum`



Building Enumerations With Python's `enum`

Building Enumerations With Python's `enum`

- Create Enumerations of Constants
- Work With Enumerations and Their Members
- Customize Enumeration Classes
- Code Practical Examples
- Explore Other Enumeration Types

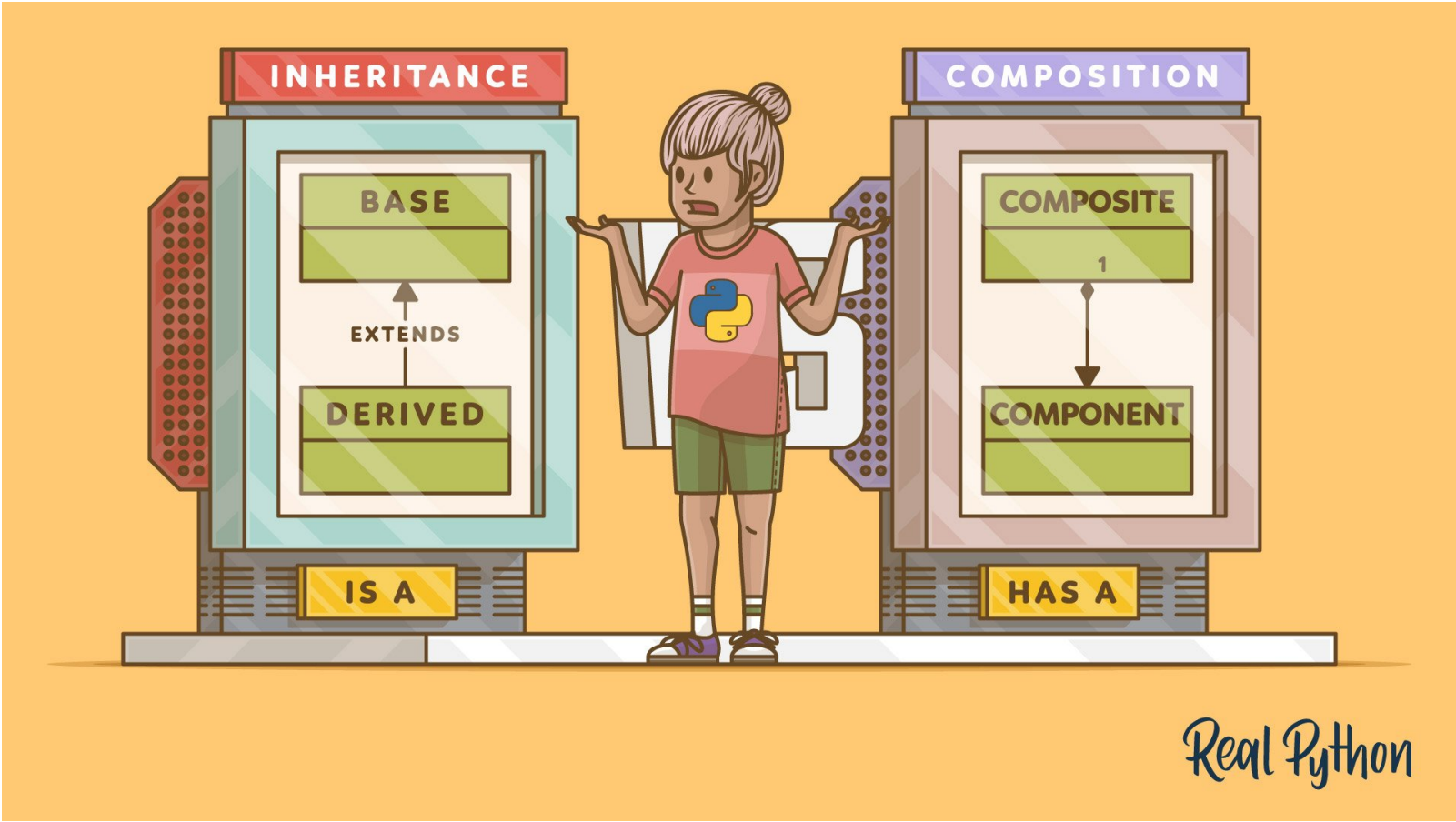
Prerequisites

- Knowledge of:
  - Object-Oriented Programming
  - Inheritance

Intro to Object-Oriented Programming (OOP) in Python

https://realpython.com/courses/intro-object-oriented-programming-oop-python/

Inheritance and Composition: A Python OOP Guide



https://realpython.com/inheritance-composition-python/

Bpython Interpreter

https://bpython-interpreter.org/

Python Versions

- Enumerations:
  - Python `>= 3.4`
- This Course:
  - Python `>= 3.11`

Let’s Get Started!

Building Enumerations With Python's `enum`

1. **Getting to Know Enumerations in Python**
2. Creating Enumerations With Python's `enum`
3. Working With Enumerations in Python
4. Extending Enumerations With New Behavior
5. Exploring Other Enumeration Classes
6. Using Enumerations: Two Practical Examples

Enumerations

- Native Data Type in Some Languages
- Sets of Named Constants:
  - Members of the Enclosing Enum
  - Accessed via the Enum
- Useful to Define Immutable, Discrete Values

Example Enumerations

- Days of the Week
- Months
- Seasons
- Cardinal Directions
- Program Status Codes
- HTTP Status Codes
- Traffic Light Colors
- Pricing Plans

Enumerations in Python

- Not a Native Data Type
- Introduced in Python 3.4 as the `enum` Module:
  - Provides the `Enum` Class
  - PEP 435 Definition:
    - “An enumeration is a set of symbolic names bound to unique, constant values. Within an enumeration, the values can be compared by identity, and the enumeration itself can be iterated over.”

Coding Without Enumerations

Enumerations in Older Python Versions

- Third-Party Libraries:
  - `enum34` - A Backport of `Enum`
  - `aenum` - Advanced Enum Library

Coding Benefits of Enumerations

- Conveniently Grouping Related Constants
- Allowing for Additional Behavior With Custom Methods
- Providing Quick and Flexible Access to Enum Members
- Enabling Direct Iteration Over Members
- Facilitating Code Completion
- Enabling Type and Error Checking
- Providing a Hub of Searchable Names
- Mitigating Spelling Mistakes

Robustness Benefits of Enumerations

- Ensuring Constant Values
- Guaranteeing Type Safety
- Improving Readability and Maintainability
- Facilitating Debugging
- Providing a Single Source of Truth

Next: Creating Enumerations With Python's `enum`

Building Enumerations With Python's `enum`

Python's `enum` Module

- Provides the `Enum` Class
- Creation:
  - Subclassing of `Enum`
  - Functional API

Creating Enumerations by Subclassing `Enum`

- General-Purpose Enumeration:
  - Iterable
  - Comparable
- Sets of Named Constants:
  - Replace Common Data Types

Representing the Days of the Week

Capitalized Member Names

“Because Enums are used to represent constants we recommend using `UPPER_CASE` names for enum members…”

https://docs.python.org/3/library/enum.html#module-enum

Differences between `Enum` and Regular Classes

- Can’t Be Instantiated
- Can’t Be Subclassed Unless the Base Enum Has No Members
- Provide a Human-Readable String Representation
- Are Iterable
- Provide Hashable Members
- Support Access Via:
  - Square Bracket Syntax - `Example["Member"]`
  - Call Syntax - `Example("Member")`
  - Dot Notation - `Example.Member`
- Don’t Allow Member Reassignments

Members Can Be of Any Type

Next: More Enumeration Creation

Building Enumerations With Python's `enum`

More Enumeration Creation

Next: Creating Enumerations With the Functional API

Building Enumerations With Python's `enum`

The Functional API

- Create Enumerations Without Class Syntax
- Call `Enum` With Appropriate Arguments

The Functional API Signature

```
Enum(
    value,
    names,
    *,
    module=None,
    qualname=None,
    type=None,
    start=1
)
```

The Functional API Argument Summary

Argument	Description	Required
`value`	Holds a String With the Name of the New Enumeration Class	Yes
`names`	Provides Names for the Enumeration Members	Yes
`module`	Takes the Name of the Module That Defines the Enumeration Class	No
`qualname`	Holds the Location of the Module That Defines the Enumeration Class	No
`type`	Holds a Class To Be Used As the First Mixin Class	No
`start`	Takes the Starting Value From the Enumeration Values Will Begin	No

The `names` Argument

- A String Containing Member Names
- An Iterable of Member Names
- An Iterable of Name-Value Pairs

The `module` and `qualname` Arguments

- Important if Enumerations are to be Pickled
- If `module` is Missing:
  - Python Will Search for It:
    - Failure will Prohibit Pickling
- If `qualname` is Not Set:
  - Python Will Set it to the Global Scope:
    - Unpickling May Fail

The `type` Argument

- Required When Providing a Mixin Class
- Can Provide New Functionality

The `start` Argument

- Customize the Initial Value
- Defaults to `1`:
  - Provides Consistency with Truthy Evaluation

Choosing a Creation Method

- Your Decision
- Depends on:
  - Taste
  - Concrete Conditions
- Functional API Needed for Dynamic Creation

Setting Custom Values

- Iterable of Name-Value Pairs

Next: Automatic Values, Aliases and Unique Values

Building Enumerations With Python's `enum`

Building Enumerations from Automatic Values

- `auto()`
- Set Automatic Values
- Consecutive Integer Values by Default

Tweaking the Behavior of `auto()`

- Override `._generate_next_value_()`

Creating Enumerations with Aliases and Unique Values

- Two or More Members with the Same Value
- Redundant Members:
  - Known as Aliases
  - Useful in Some Situations

Current Progress

- What Enumerations Are - ✅
- When to Use Enumerations - ✅
- Benefits of Enumerations - ✅
- Creation via:
  - Inheritance - ✅
  - Calling `Enum` - ✅

Next: Working With Enumerations in Python

Building Enumerations With Python's `enum`

Accessing Enumeration Members

- Fundamental Operation
- Three Access Methods

Using the `.name` and `.value` Attributes

- Enumeration Members are Instances of Containing Class
- `.name` Attribute:
  - Automatically Provided
  - Contains Name as a String
- `.value` Attribute:
  - Contains Value

Iterating Through Enumerations

- Enumerations are Iterable by Default

Next: Using Enumerations in `if` and `match` Statements

Building Enumerations With Python's `enum`

Using Enumerations in `if` and `match` Statements

- `if ... elif`
- `match ... case`
- Different Action Depending on Conditions

A Traffic Light Example

A Note on `if` and `match` Statements

- Work Well Initially
- Don’t Scale Well:
  - New Members Mean New Statements

Next: Comparing and Sorting Enumerations

Building Enumerations With Python's `enum`

Comparing Enumerations

- Supported Default Comparison Operators:
  - Identity - `is` and `is not`
  - Equality - `==` and `!=`

The Identity Comparison

- Each Enum is a Singleton Instance of Enumeration Class
- Fast Identity Comparison

Current Progress

- Create Enumerations - ✓
- Use Enumerations - ✓
- Default Enumerations - ✓
- Custom Behavior:
  - Add Methods
  - Mixin Classes

Next: Extending Enumerations With New Behavior

Building Enumerations With Python's `enum`

1. Getting to Know Enumerations in Python
2. Creating Enumerations With Python's `enum`
3. Working With Enumerations in Python
4. **Extending Enumerations With New Behavior**
5. Exploring Other Enumeration Classes
6. Using Enumerations: Two Practical Examples

Adding and Tweaking Member Methods

- Adding New Methods:
  - Methods
  - Special Methods

Mixing Enumerations with Other Types

- Python Supports Multiple Inheritance
- Useful to Inherit Functionality from Several Classes
- Mixin Classes are Common Practice:
  - Provide Functionality for Other Classes
  - Add Mixin Classes to a List of Parent Classes

Integer Enumerations are Common

- `enum` Provides `IntEnum`
- Covered Later in this Course

Signature When Using Mixin Classes

```
class EnumName([mixin_type, ...], [data_type,] enum_type):
    # Members go here...
```

Next: Exploring Other Enumeration Classes

Building Enumerations With Python's `enum`

1. Getting to Know Enumerations in Python
2. Creating Enumerations With Python's `enum`
3. Working With Enumerations in Python
4. Extending Enumerations With New Behavior
5. **Exploring Other Enumeration Classes**
6. Using Enumerations: Two Practical Examples

Exploring Other Enumeration Classes

- `Enum` - Base Class for Enumerations
- `IntEnum` - Enumerations that are Subclasses of `int`
- `StrEnum` - Enumerations that are Subclasses of `str`
- `IntFlag` - `int` Subclass, Can Be Combined with Bitwise Operators
- `Flag` - Can Be Combined with Bitwise Operators

Building Integer Enumerations: `IntEnum`

- Created to Cover a Common Use Case
- Use When Integer Behaviour is Desirable

Building String Enumerations: `StrEnum`

- Python `>= 3.11`
- Support for Common String Operations

Creating Integer Flags: `IntFlag`

- Base Class that Supports Bitwise Operators
- Return an Object that is Member of `IntFlag`

Enumeration Advantages

- Improve:
  - Readability
  - Organization
- Group Similar Values Together
- Replace:
  - Strings
  - Numbers
  - Other Values

Next: Using Enumerations: Two Practical Examples

Building Enumerations With Python's `enum`

1. Getting to Know Enumerations in Python
2. Creating Enumerations With Python's `enum`
3. Working With Enumerations in Python
4. Extending Enumerations With New Behavior
5. Exploring Other Enumeration Classes
6. **Using Enumerations: Two Practical Examples**

Replacing Magic Numbers

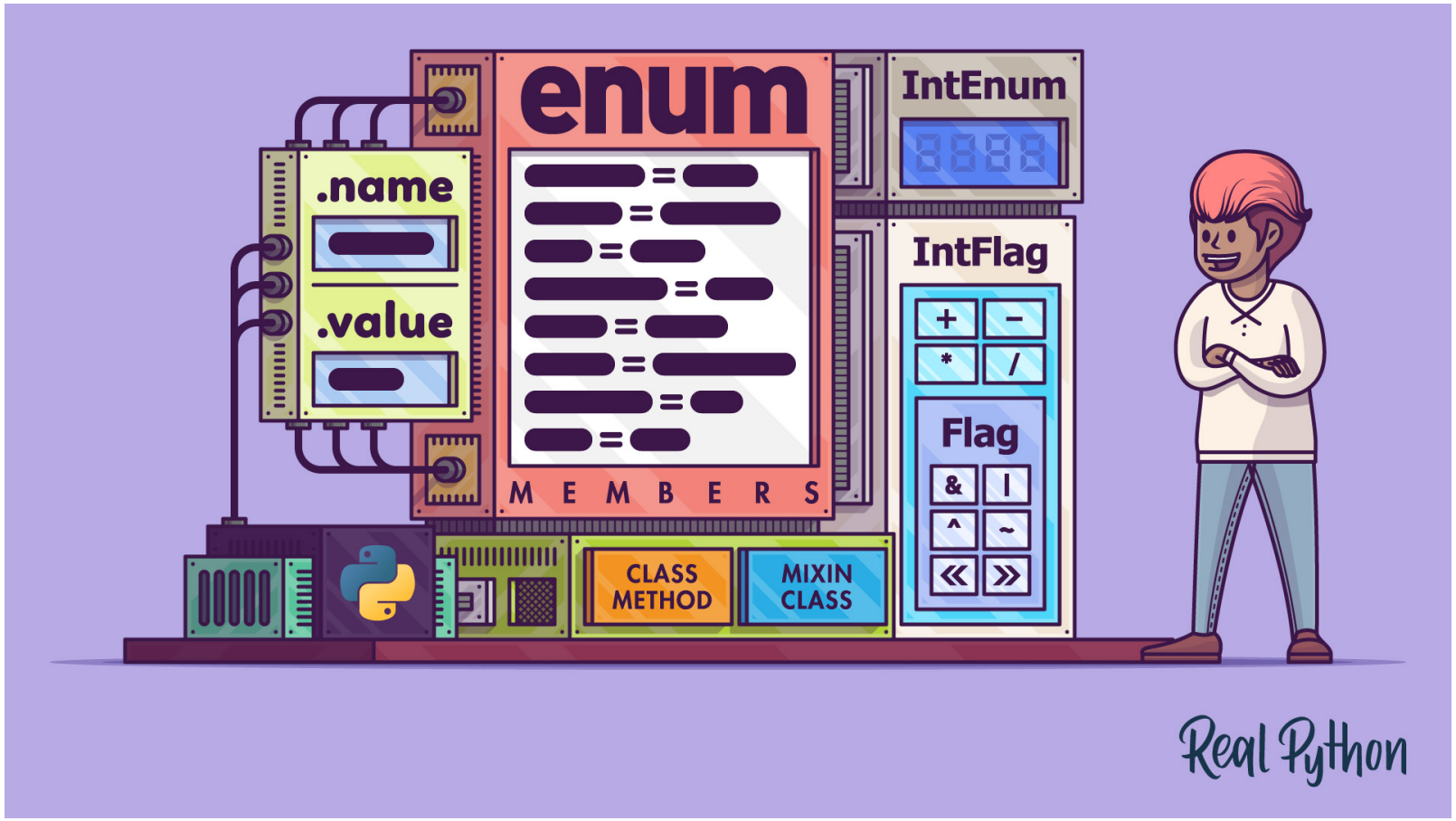
- Replacing:
  - HTTP Status Codes
  - Computer Ports
  - Exit Codes
- Group Constants Together
- Assign Meaningful Names

Creating a State Machine

- States of a Given System
- Common Design Pattern
- Disk Player Simulator Example

Next: Summary

Building Enumerations With Python's `enum` : Summary



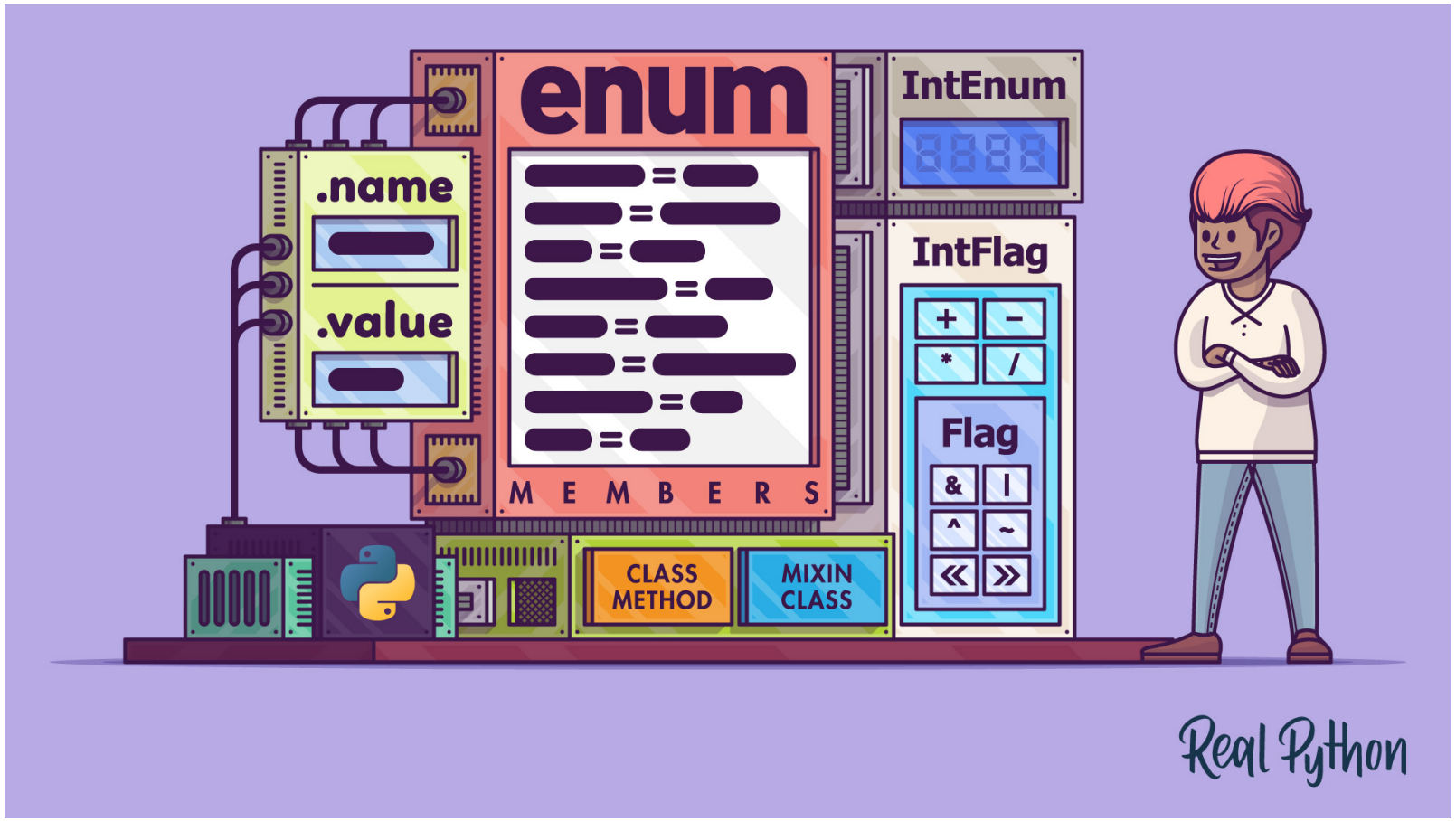
Summary

- Create and Use Enumerations
- Common Data Type
- Used for Grouping Sets of Constants
- Provided via `enum` Module in Python `>= 3.4`

Summary

- Create Enumerations of Constants using `Enum`
- Work With Enumerations and Their Members
- Add Functionality to Enumeration Classes
- Code Practical Examples
- Explore Other Enumeration Types:
  - `IntEnum`
  - `StrEnum`
  - `IntFlag`
  - `Flag`

Summary


enum
.name
.value
MEMBERS
IntEnum
IntFlag
Flag
CLASS METHOD
MIXIN CLASS
Real Python