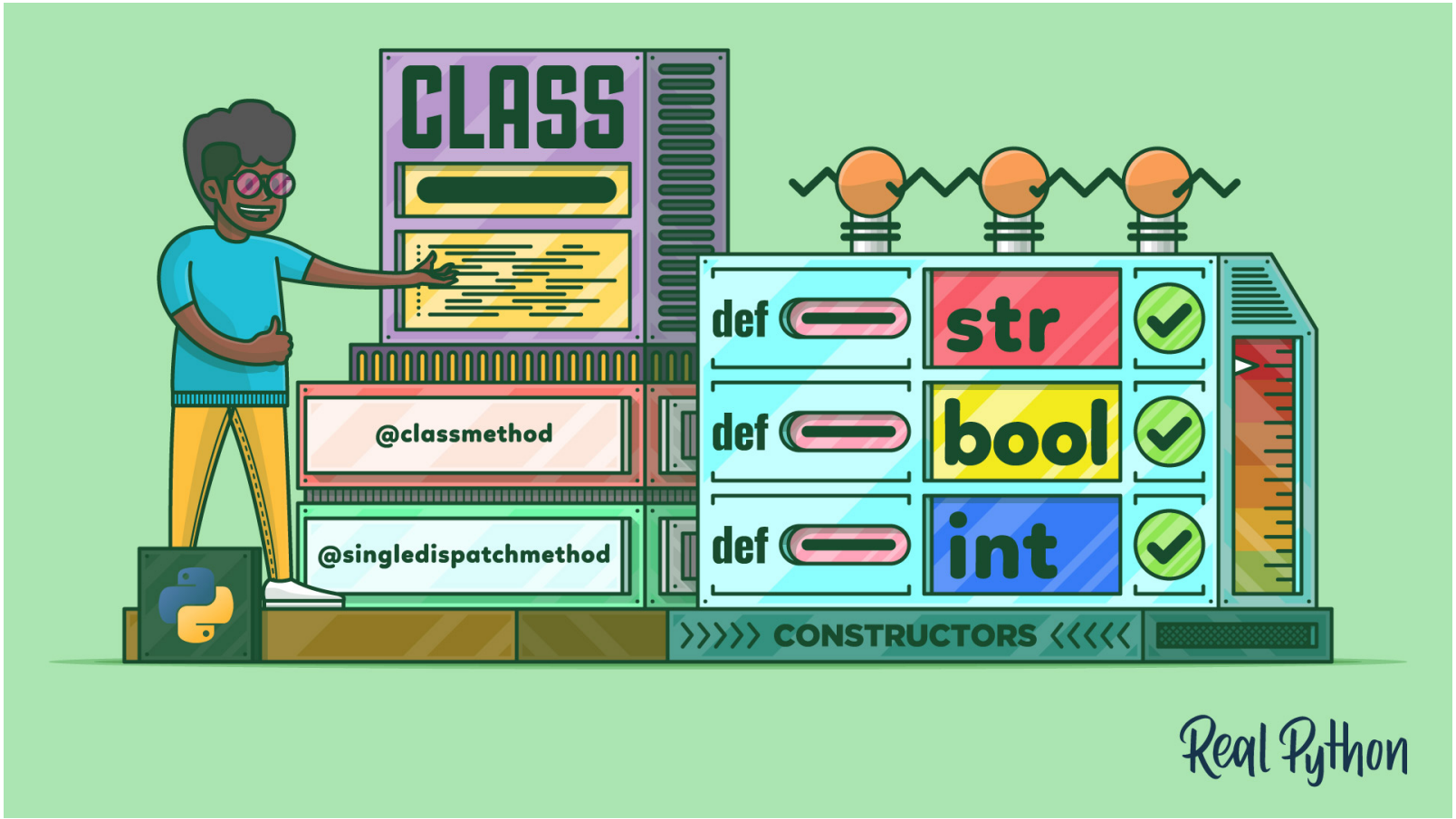
Providing Multiple Class Constructors



Providing Multiple Class Constructors

Providing Multiple Class Constructors

- Simulate Multiple Constructors with
  - Optional Arguments
  - Type Checking
- Write Multiple Constructors Using `@classmethod`
- Overload Class Constructors Using `@singledispatchmethod`

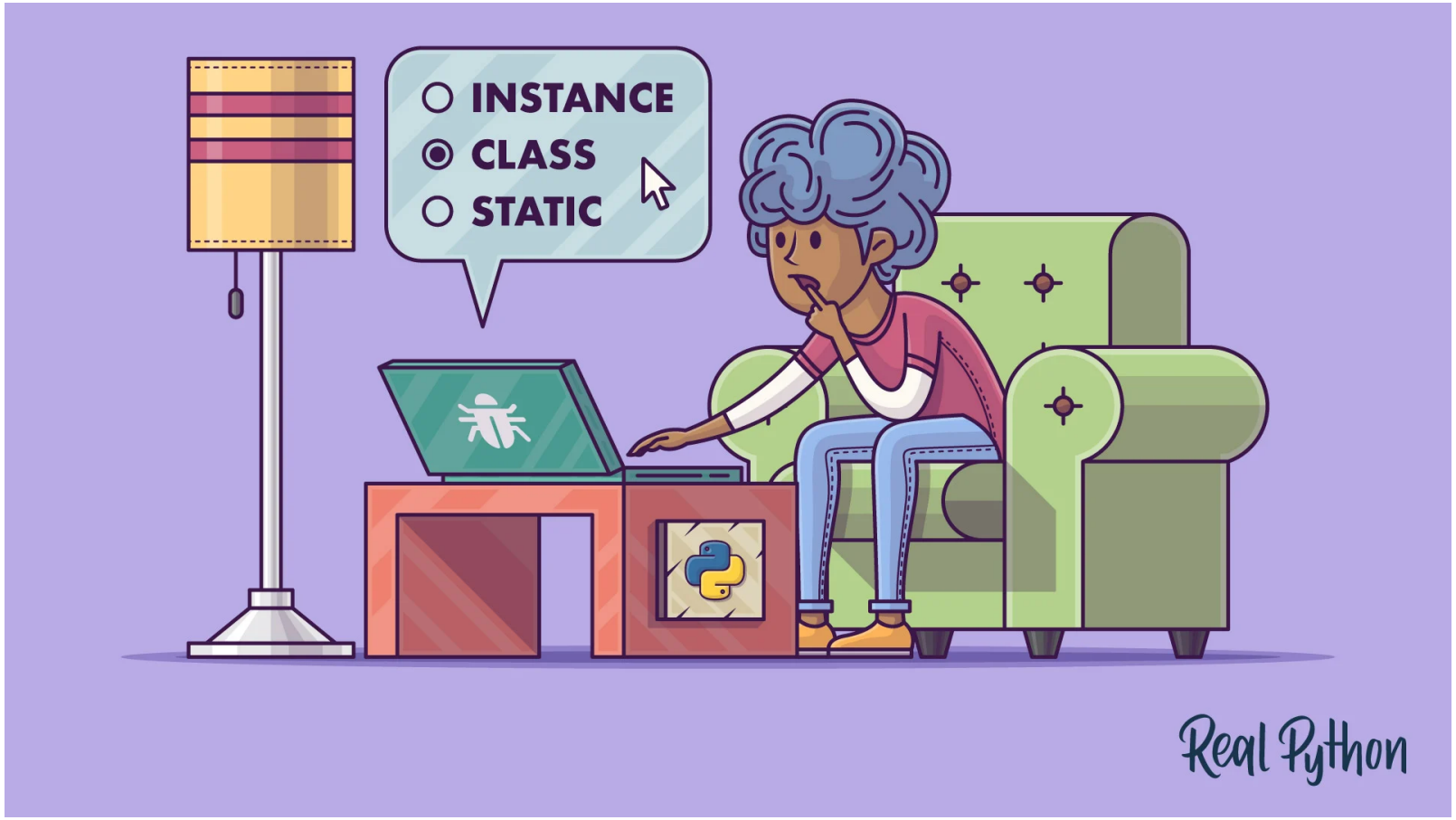
Prerequisites

- Object-Oriented Programming
- Class Methods
- Decorators

Intro to Object-Oriented Programming in Python

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

OOP Method Types in Python



https://realpython.com/courses/python-method-types/

Python Decorators 101

https://realpython.com/courses/python-decorators-101/

Bpython Interpreter

https://bpython-interpreter.org/

Let’s Get Started!

Providing Multiple Class Constructors

1. **Instantiating Classes in Python**
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
4. Using `@classmethod` for Multiple Class Constructors
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors

Instantiating Classes in Python

- Object-Oriented Programming
- **Classes** - The Blueprint for Objects
- **Instances** - The Individual Objects

Special Methods

- Make Iterable - `__iter__()`
- Provide String Representation - `__repr__()`
- Initialize Instance Attributes - `__init__()`
- … and Many More

`__init__(self, name)`

- Instance Initializer
  - Initialize Instance Attributes
- First Argument: `self`
  - Holds Current Object
  - Passed Implicitly In Method Call
- Second Argument: `name`
  - In This Case, Passes a String

Instantiation Process

- Create a New Instance - `__new__()`
- Initialize the Instance - `__init__()`

New Instance Creation With `__new__()`

- Creates New Instances
- Often Called Class Constructor
- More Accurately Called:
  - Instance Creator
  - Object Creator

New Instance Creation With `__new__()`

- Takes Underlying Class as First Argument
- Returns a New Object
  - Typically the Input Class - handled by `__init__()`
  - Can Be an Instance of Another Class - `__init__()` Omitted

Object Class

- Provides Default Implementations of:
  - `__new__()`
  - `__init__()`
- `__new__()` Rarely Needs Overriding

Instantiation Process Summary

- Call to Class with Appropriate Arguments
- Object Creation with `__new__()`
- Object Initialization with `__init__()`

Next: Defining Multiple Class Constructors

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. **Defining Multiple Class Constructors**
3. Simulating Multiple Class Constructors
4. Using `@classmethod` for Multiple Class Constructors
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors

Defining Multiple Class Constructors

- Arguments of Different Data Types
- Different Number of Arguments

Method Overloading

- Supported by C++, C# and Java
- Creation of Multiple Methods With the Same Name
- Language Selects Appropriate Method to Use
- Method Can Perform Different Tasks Depending on Context
- Not Supported Directly by Python

Method Names in Python

- Stored in an Internal Dictonary - `__dict__`
- Holds the Class Namespace
- Dictionaries Cannot Support Repeated Keys
- Multiple Methods With the Same Name Are Not Possible
- Only Last Implementation of a Given Name is Stored

Multiple Dispatch

- Several Different Implementations of Same Method
- Dynamically Dispatched Depending on Argument
  - Type
  - Other Characteristics

Next: Simulating Multiple Class Constructors

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. **Simulating Multiple Class Constructors**
   3.1 Using Optional Argument Values in `__init__()`
   3.2 Checking Argument Types in `__init__()`
4. Using `@classmethod` for Multiple Class Constructors
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors

Simulating Multiple Class Constructors

- Provide `__init__()` With:
  - Optional Arguments
  - Default Values
- Check Data Types Passed to `__init__()`
  - Change Behaviour as a Result

Next: Using Optional Argument Values in `__init__()`

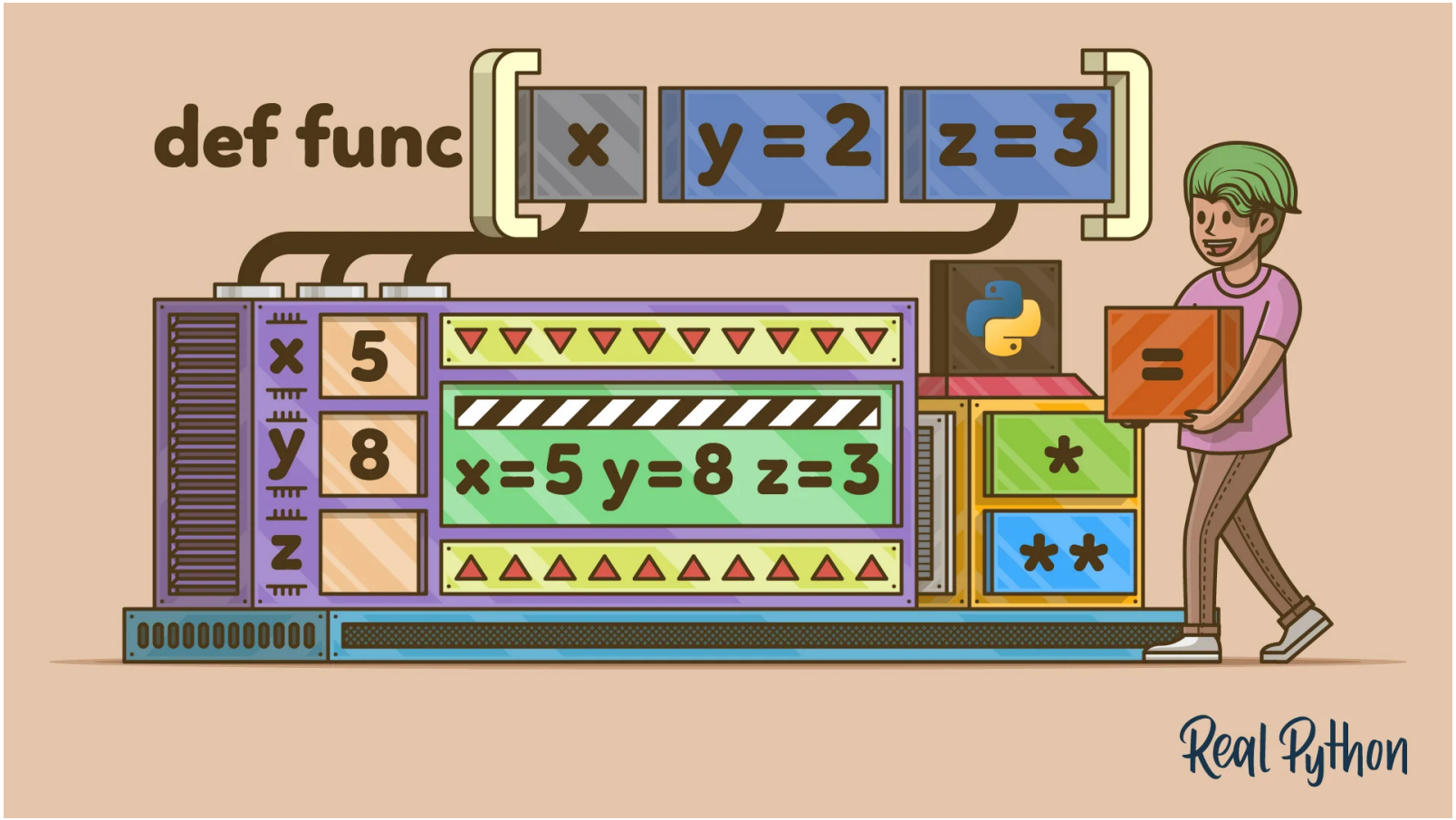
Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
   - 3.1 **Using Optional Argument Values in `__init__()`**
   - 3.2 Checking Argument Types in `__init__()`
4. Using `@classmethod` for Multiple Class Constructors
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors

Using Optional Argument Values in `__init__()`

- Optional Arguments
- Appropriate Default Values
- Undefined Number of:
  - Positional Arguments
  - Keyword Arguments

Defining Python Functions With Optional Arguments



https://realpython.com/courses/defining-python-functions-with-optional-arguments/

Example: `CumulativePowerFactory`

- Create Callable Objects
- Compute Specific Powers
- Stream of Numbers as Input
- Total Sum Of Powers Stored
- Initial Sum Argument

Optional Arguments Are Clean and Pythonic

Next: Checking Argument Types in `__init__()`

Providing Multiple Class Constructors

Checking Argument Types in `__init__()`

`Person` Class

- Accept a Birth Date
- Represented as a `date` Object
- Provided as a `date` or String

This Technique Does Not Scale Well

- Multiple Argument Inspection Becomes Complex
- Considered an Anti-Pattern

PEP 443: Single-Dispatch Generic Functions

“…it is currently a common anti-pattern for Python code to inspect the types of received arguments, in order to decide what to do with the objects.”

https://peps.python.org/pep-0443/

Next: Using `@classmethod` for Multiple Class Constructors

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
4. **Using `@classmethod` for Multiple Class Constructors**
   4.1 Constructing a Circle From Its Diameter
   4.2 Building a Polar Point From Cartesian Coordinates
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors

Using `@classmethod` for Multiple Class Constructors

`@classmethod`

- Doesn’t Take an Instance - `self` - as an Argument
- Takes Current Class - `cls` - Instead

Using `@classmethod` as a Constructor

- Doesn’t Need an Instance to Be Called
- Add Multiple Explicit Constructors
- The `@classmethod` Returns the Instance
- Pythonic
- Known as an Alternative Constructor

Using `@classmethod` as a Constructor

- Replacement of Fine-Tuning `__init__()`
- Full Control of:
  - Creation
  - Initialization

Next: Constructing a Circle From Its Diameter

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
4. Using `@classmethod` for Multiple Class Constructors
   - 4.1 **Constructing a Circle From Its Diameter**
   - 4.2 Building a Polar Point From Cartesian Coordinates
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors

Constructing a Circle From Its Diameter

Instantiating a Circle From Its Diameter

Instantiating a Circle From Its Diameter

```
new_circle = Circle(diameter / 2)
```

Using `@classmethod` for Multiple Class Constructors

- Commonly Used
- Allows Precise Name Selection
- Makes Code Readable and Maintainable

Next: Building a Polar Point From Cartesian Coordinates

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
4. Using `@classmethod` for Multiple Class Constructors
   4.1 Constructing a Circle From Its Diameter
   4.2 **Building a Polar Point From Cartesian Coordinates**
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors

Building a Polar Point From Cartesian Coordinates

`Polar` Point Class

- Points Represented as `distance` and `angle`
- Cartesian Representation Desirable using `x` and `y`

Next: Exploring Multiple Class Constructors in Existing Classes

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
4. Using `@classmethod` for Multiple Class Constructors
5. **Exploring Multiple Class Constructors in Existing Classes**
6. Using `@singledispatchmethod` for Multiple Class Constructors

Exploring Multiple Class Constructors in Existing Classes

- `dict`
- `datetime.date`
- `pathlib.Path`

Dictionaries

- Fundamental Data Type
- Present in All Python Code
- A Cornerstone of CPython
- Multiple Definition Methods
  - `{'key': 'value'}`
  - `dict()`
  - `.fromkeys()`

Dictionary `.fromkeys()`

- Takes an Iterable of Keys
- Optional Value
  - Defaults to `None`
  - Value For All Keys In Dictionary

Using Dictionary `.fromkeys()`

Other Classes With a `.fromkeys()` Method

- `OrderedDict`
- `defaultdict`
- `UserDict`

`UserDict` Source Code

```
    @classmethod
    def fromkeys(cls, iterable, value=None):
        d = cls()
        for key in iterable:
            d[key] = value
        return d
```

`UserDict` Source Code

```
@classmethod
def fromkeys(cls, iterable, value=None):
    d = cls()
    for key in iterable:
        d[key] = value
    return d
```

`UserDict` Source Code

```
@classmethod
def fromkeys(cls, iterable, value=None):
    d = cls()
    for key in iterable:
        d[key] = value
    return d
```

`UserDict` Source Code

```
@classmethod
def fromkeys(cls, iterable, value=None):
    d = cls()
    for key in iterable:
        d[key] = value
    return d
```

`UserDict` Source Code

```
@classmethod
def fromkeys(cls, iterable, value=None):
    d = cls()
    for key in iterable:
        d[key] = value
    return d
```

`UserDict` Source Code

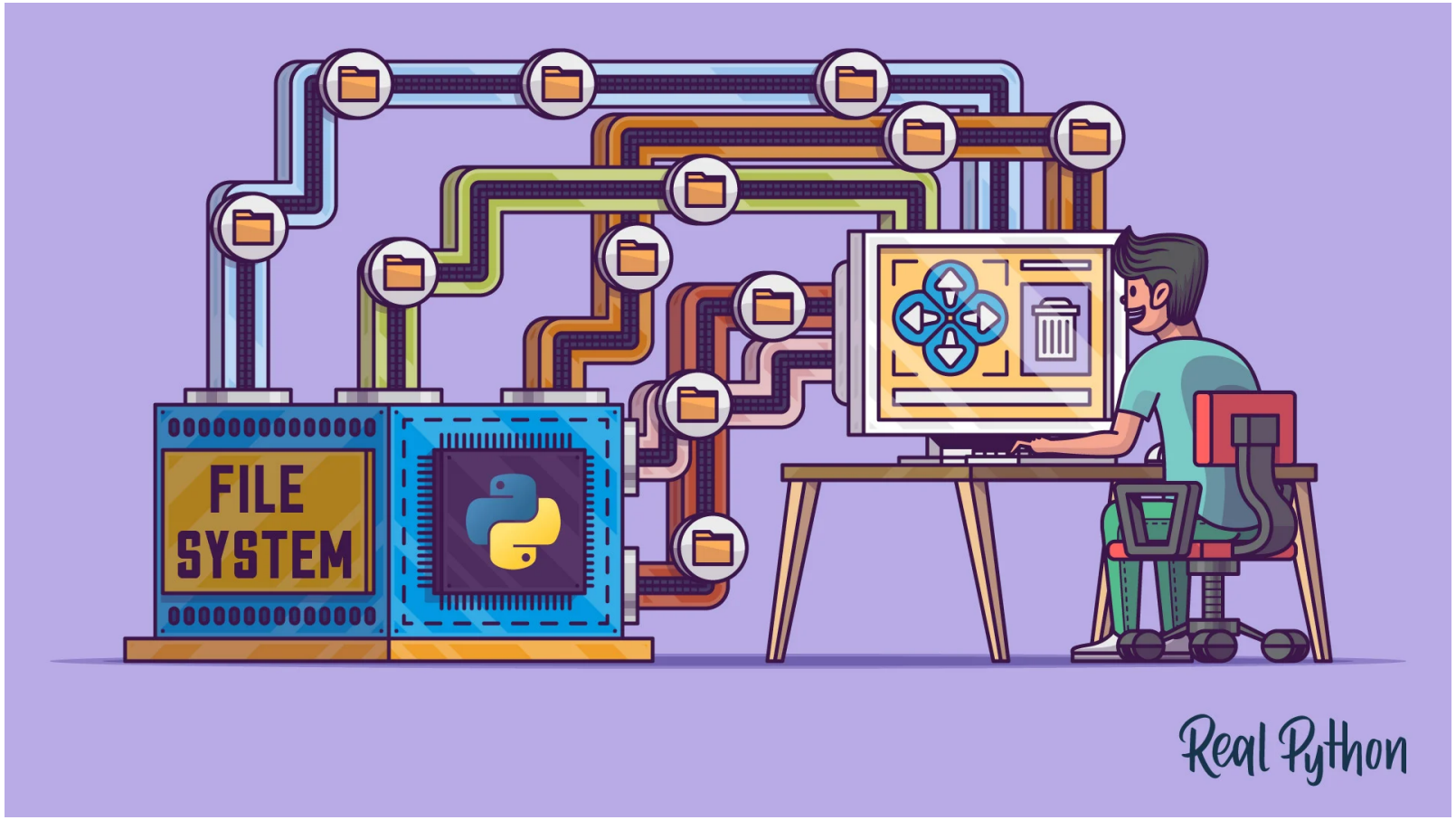
```
@classmethod
def fromkeys(cls, iterable, value=None):
    d = cls()
    for key in iterable:
        d[key] = value
    return d
```

Creating `datetime.date` Objects

- `.today()`
- `.fromtimestamp()`
- `.fromordinal()`
- `.fromisoformat()`

Finding Your Path to Home

Python 3's `pathlib` Module: Taming the File System



https://realpython.com/python-pathlib/

Pathlib's `Path` Class

- Works Cross-Platform
- Provides Multiple Constructors:
  - `.home()`
  - `.cwd()`

```
Path.home()
```

`Path.cwd()`

- `Path` of **C**urrent **W**orking **D**irectory

Next: Using `@singledispatchmethod` for Multiple Class Constructors

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
4. Using `@classmethod` for Multiple Class Constructors
5. Exploring Multiple Class Constructors in Existing Classes
6. **Using `@singledispatchmethod` for Multiple Class Constructors**
   6.1 A Real-World Example of a Single-Dispatch Method

Single-Dispatch Generic Function

- Add Multiple Constructors
- Run Selectively Depending on Argument

Single-Dispatch Generic Function

- Multiple Functions
- Same Operation
- Different Data Types
- Single Dispatch Algorithm
- Python 3.8 or Greater
  - `@singledispatch` or `@singledispatchmethod`

Python Implementation of Single-Dispatch Functions

- Defined in PEP 443
- Part of `functools` Module
- Function Selected by First Argument
- Method Selected by Argument After `self`

Single-Dispatch Method Demo

- Define Base Method
- Decorate with `@singledispatchmethod`
- Create Alternative Implementations
- Register with `@base_method.register`

Next: A Real-World Example of Single-Dispatch Method

Providing Multiple Class Constructors

1. Instantiating Classes in Python
2. Defining Multiple Class Constructors
3. Simulating Multiple Class Constructors
4. Using `@classmethod` for Multiple Class Constructors
5. Exploring Multiple Class Constructors in Existing Classes
6. Using `@singledispatchmethod` for Multiple Class Constructors
   - 6.1 **A Real-World Example of a Single-Dispatch Method**

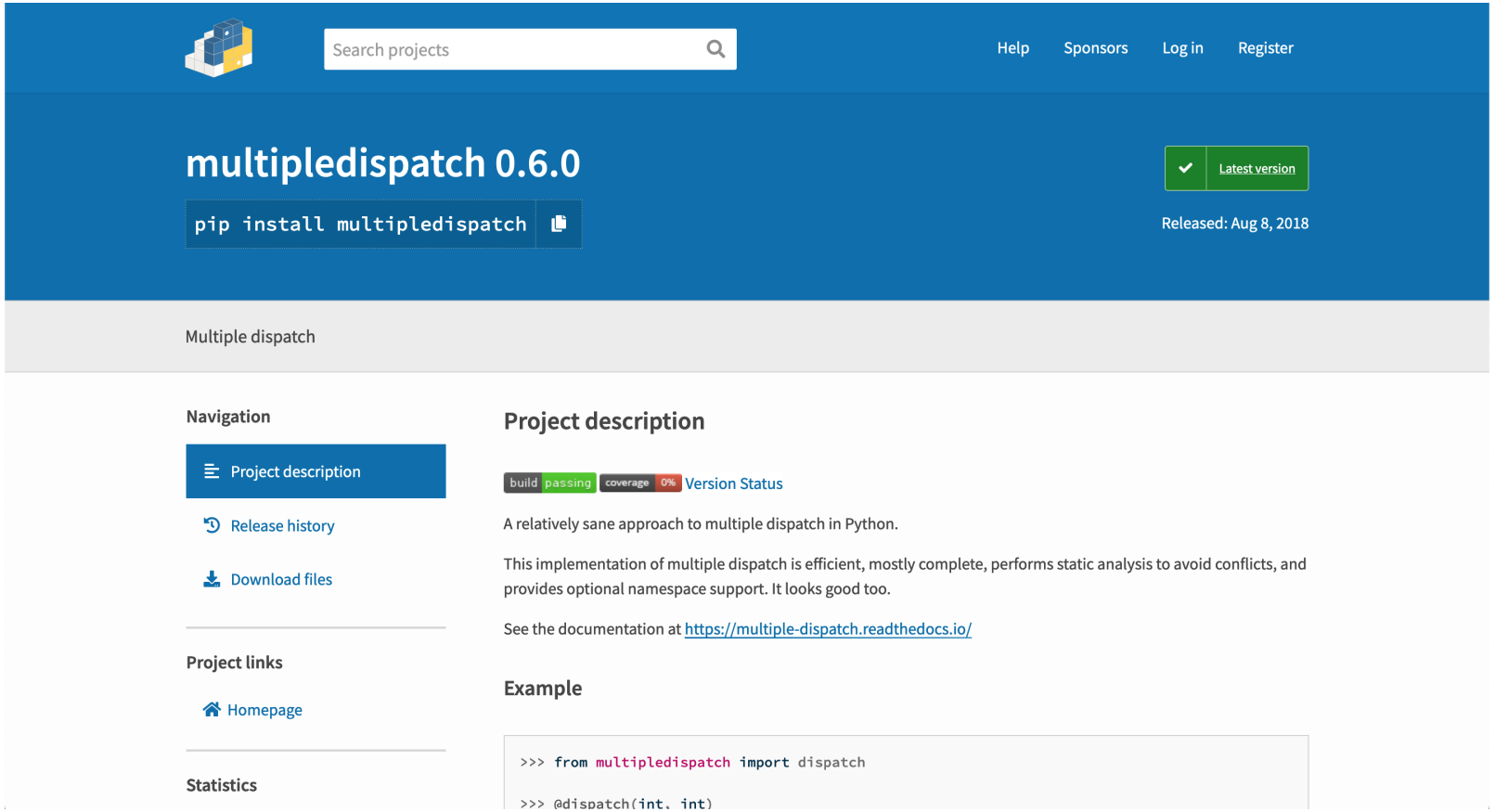
`Person` Class New Features

- Computation of Age Based on Birth Date
- Helper Class To Handle Birth Date and Age

Single-Dispatch Function Limitation

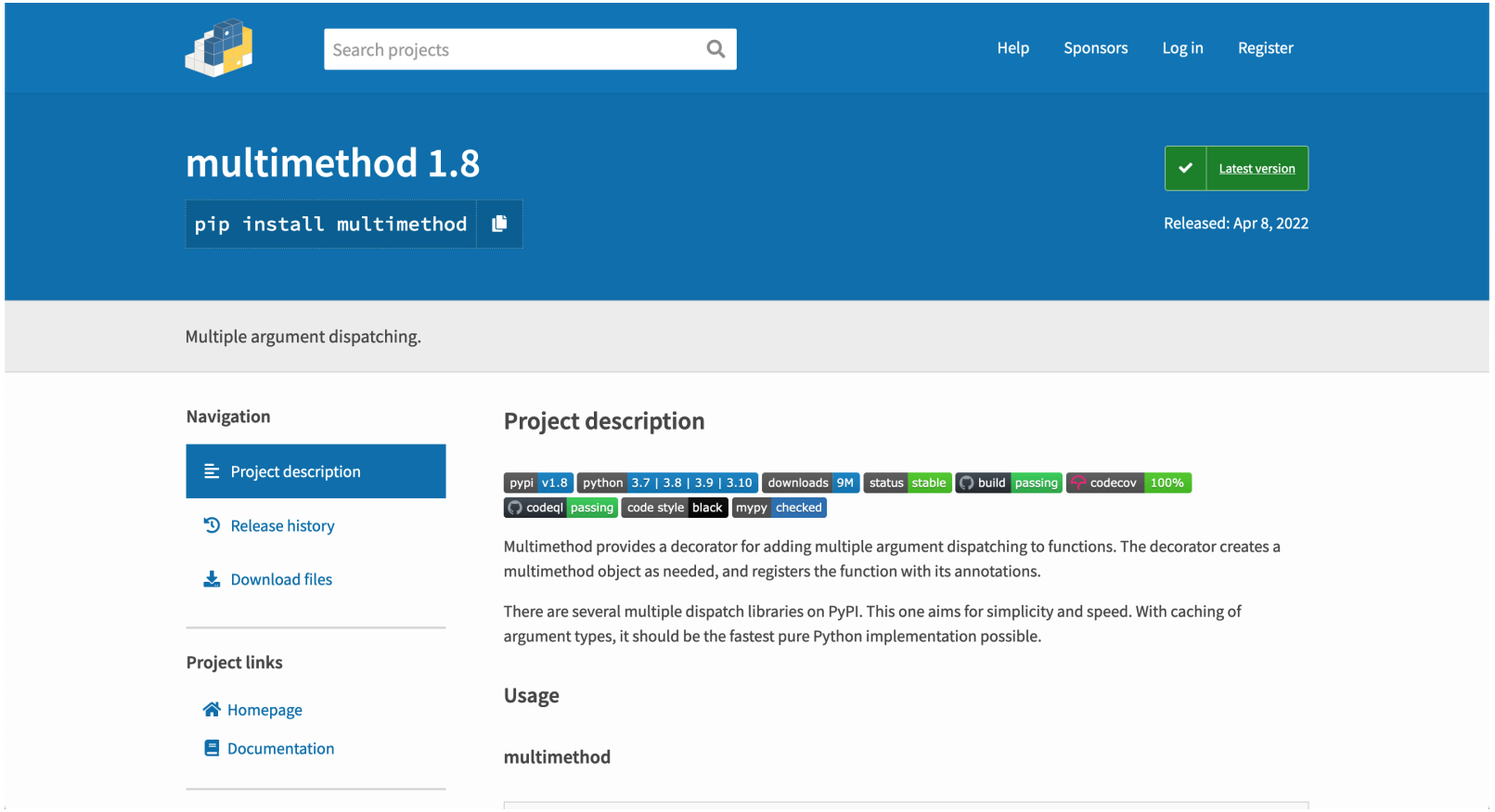
- Relies on Single Argument

multipledispatch



https://pypi.org/project/multipledispatch/

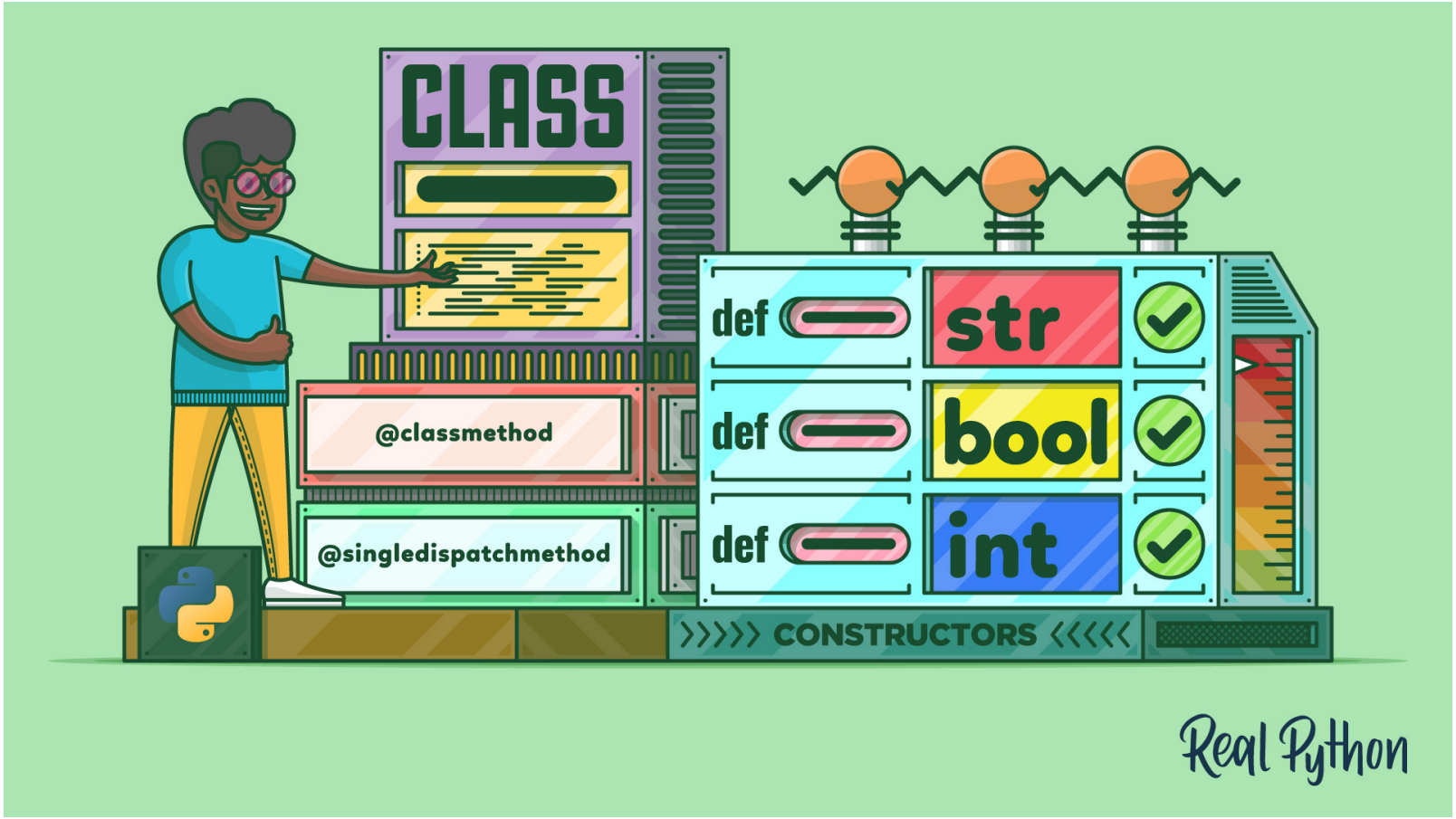
multimethod



https://pypi.org/project/multimethod/

Next: Summary

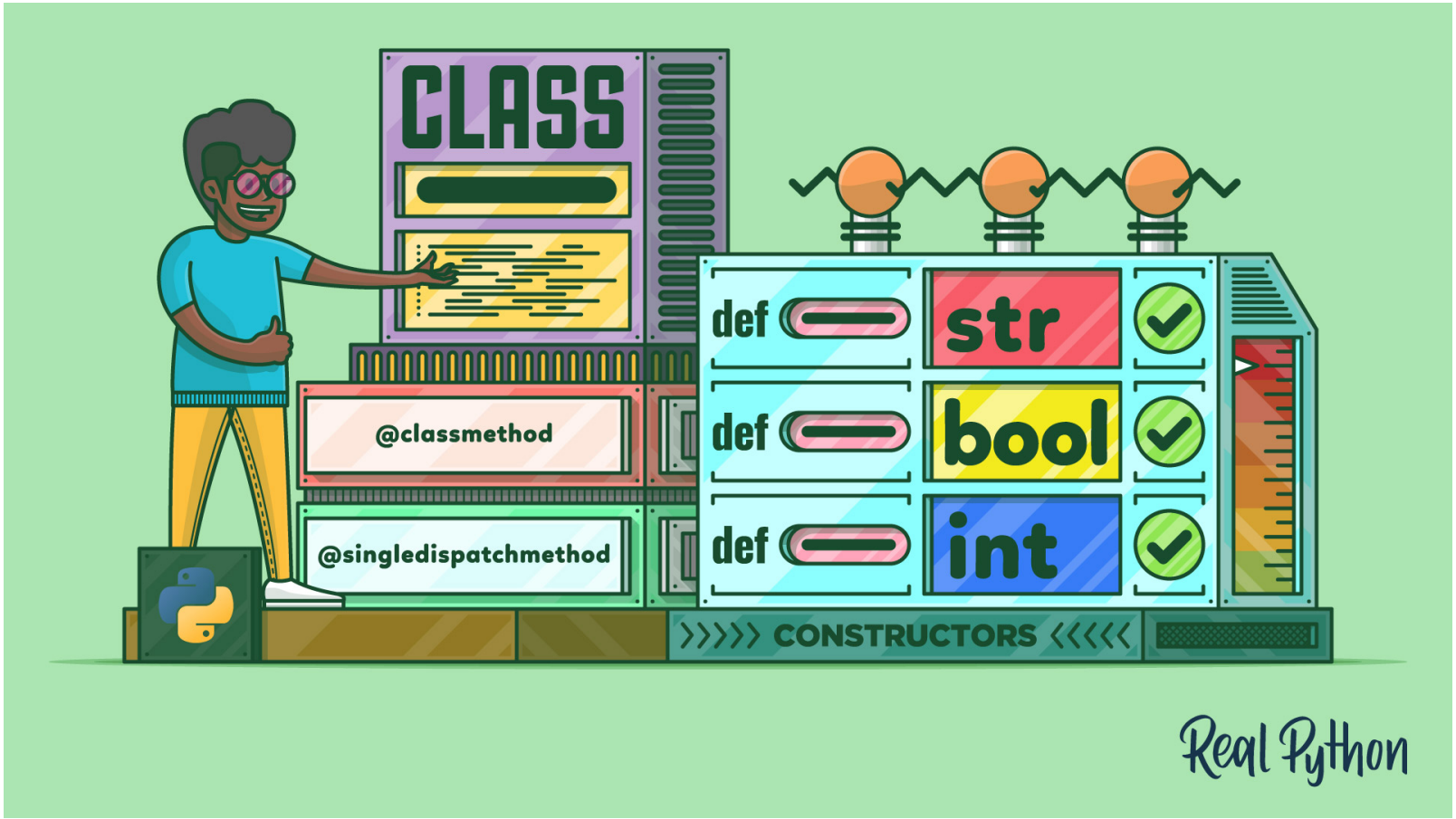
Providing Multiple Class Constructors : Summary



Summary

- Simulate Multiple Constructors
  - Optional Arguments
  - Type Checking
- Write Multiple Constructors Using the Built-in `@classmethod`
- Overload Class Constructors Using `@singledispatchmethod`

Summary


CLASS
@classmethod
@singledispatchmethod
def
str
def
bool
def
int
CONSTRUCTORS
Real Python