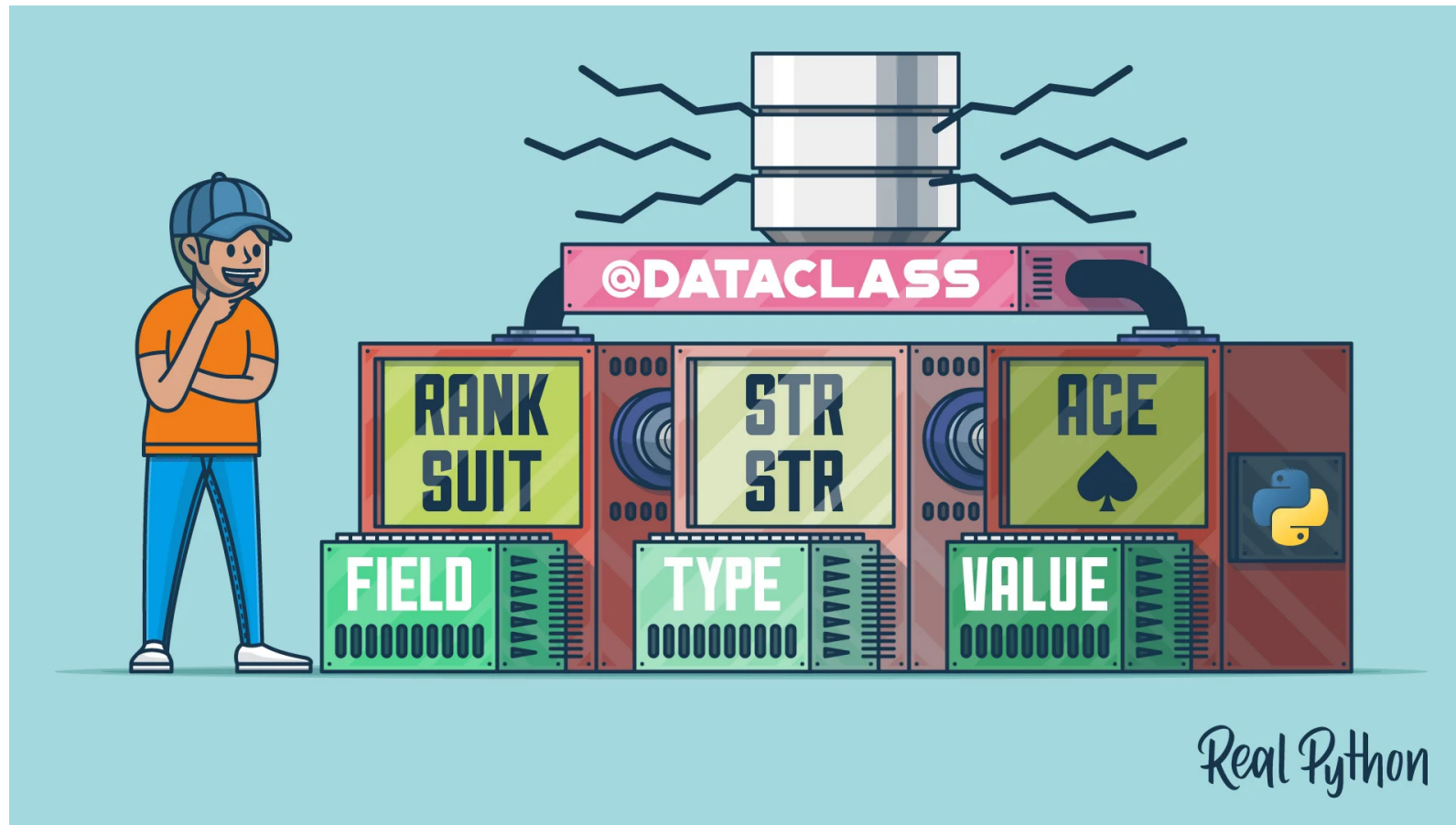


Python Data Classes



Python Data Classes

Python Data Classes

- Representation
- Comparison
- Adding Default Values
- Ordering
- Immutable Data
- Inheritance

Next: Comparison to Standard Classes

Python Data Classes

- ▶ 1. **Comparison to Standard Classes**
- 2. Alternatives to Data Classes
- 3. Basic Data Classes
- 4. More Flexible Data Classes
- 5. Immutable Data Classes
- 6. Inheritance
- 7. Optimizing Data Classes

System Setup

- Python 3.9
- Python ≥ 3.7

Bpython **REPL**

<https://bpython-interpreter.org/>

@dataclass Decorator

```
@dataclass
class DataClass:
    variable: type
    variable: type
```

Dataclass Methods

- `__repr__()` - Representation
- `__eq__()` - Simple Comparisons

Next: Alternatives to Data Classes

Python Data Classes

1. Comparison to Standard Classes

▶ 2. **Alternatives to Data Classes**

3. Basic Data Classes

4. More Flexible Data Classes

5. Immutable Data Classes

6. Inheritance

7. Optimizing Data Classes

Alternatives to Data Classes

- Tuple
- Dictionary

Problems with Tuples and Dictionaries

- Remembering a **Variable** is a Specific Data Type
- Order Attributes for Tuples

Problems with Tuples and Dictionaries

- Remembering a **Variable** is a Specific Data Type
- Order Attributes for Tuples

```
queen_of_hearts_tuple = ('Q', 'Hearts')  
ace_of_spades_tuple = ('Spades', 'A')
```

Problems with Tuples and Dictionaries

- Remembering a **Variable** is a Specific Data Type
- Order Attributes for Tuples
- Consistent Attribute Names for Dictionaries

Problems with Tuples and Dictionaries

- Remembering a **Variable** is a Specific Data Type
- Order Attributes for Tuples
- Consistent Attribute Names for Dictionaries

```
queen_of_hearts_dict = {'rank': 'Q', 'suit': 'Hearts'}  
ace_of_spades_dict = {'value': 'A', 'suit': 'Spades'}
```

The Attrs Project



<https://www.attrs.org>

The Attrs Project

- Supports Converters and Validators
- Mature Project
- Supports Old Python Versions
- **Not** in Standard Library

Other Alternatives

- `tuple`
- `dict`
- `collections.NamedTuple`
- `attr` - <https://www.attrs.org/>
- `typing.NamedTuple`
- `attrdict` - <https://pypi.org/project/attrdict/>
- `plumber` - <https://pypi.org/project/plumber/>
- `fields` - <https://pypi.org/project/fields/>

Dataclasses are Not a Universal Solution

- Compatibility with Specific API
- Functionality Not Present

Next: Basic Data Classes

Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
- ▶ 3. **Basic Data Classes**
4. More Flexible Data Classes
5. Immutable Data Classes
6. Inheritance
7. Optimizing Data Classes

Basic Data Classes

Dataclass Included Methods

- `__init__()` - Initialization
- `__repr__()` - Representation
- `__eq__()` - Simple Comparisons

Default Values

Type Hint Definitions

```
@dataclass  
class DataClass:
```

Type Hint Definitions

```
@dataclass  
class DataClass:  
    name: str
```

Type Hints are Mandatory in Dataclasses

Adding Methods

Adding Methods

```
@dataclass
class DataClass:
    name: str

    def method(self):
        return True
```

Calculating Position on Earth's Surface

Calculating Position on Earth's Surface

$$d = 2r \arcsin \left(\sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)} \right)$$

https://en.wikipedia.org/wiki/Haversine_formula

Next: More Flexible Data Classes

Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
3. Basic Data Classes
- ▶ 4. **More Flexible Data Classes**
5. Immutable Data Classes
6. Inheritance
7. Optimizing Data Classes

More Flexible Data Classes

- `@dataclass` Parameters
- `field`

Next: Advanced Default Values

Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
3. Basic Data Classes
4. More Flexible Data Classes
- ▶ 4.1 **Advanced Default Values**
 - 4.2 Representation
 - 4.3 Comparison
5. Immutable Data Classes
6. Inheritance
7. Optimizing Data Classes

Advanced Default Values

Unicode Symbols

Unicode Card Symbols

Symbol	Unicode	Name
♠	U + 2660	BLACK SPADE SUIT
♦	U + 2662	WHITE DIAMOND SUIT
♥	U + 2661	WHITE HEART SUIT
♣	U + 2663	BLACK CLUB SUIT

Unicode Input

Article [Talk](#) [Read](#) [Edit](#) [View history](#) [More](#)

Unicode input

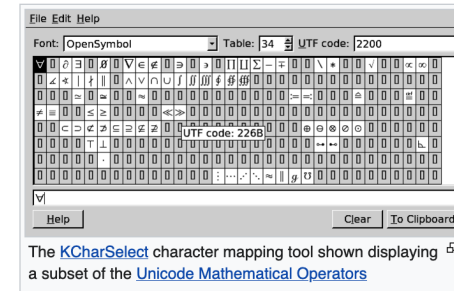
From Wikipedia, the free encyclopedia

Unicode input is the insertion of a specific [Unicode character](#) on a computer by a [user](#); it is a common way to input characters not directly supported by a physical [keyboard](#). Unicode characters can be produced either by selecting them from a display or by typing a certain sequence of keys on a physical keyboard. In addition, a character produced by one of these methods in one web page or document can be [copied](#) into another. In contrast to [ASCII](#)'s 96 element [character set](#) (which it contains), Unicode encodes hundreds of thousands of [graphemes](#) (characters) from almost all of the world's written languages and many other signs and symbols besides. ^[1]^{*[better source needed]*}

A Unicode input system must provide for a large repertoire of characters, ideally all valid Unicode code points. This is different from a [keyboard layout](#) which defines keys and their combinations only for a limited number of characters appropriate for a certain [locale](#).

Contents [\[hide\]](#)

- [1 Unicode numbers](#)
- [2 Availability](#)
- [3 Selection from a screen](#)
- [4 Decimal input](#)
- [5 Hexadecimal input](#)
 - [5.1 In Microsoft Windows](#)
 - [5.2 In MacOS](#)
 - [5.3 In X11 \(Linux and other Unix variants including Chrome OS\)](#)
 - [5.4 In platform-independent applications](#)
- [6 HTML](#)
- [7 See also](#)
- [8 Notes](#)
- [9 References](#)



https://en.wikipedia.org/wiki/Unicode_input

Unicode Printing In Python

```
>>> print("\u2660")
```

♠

```
>>> print("\N{WHITE HEART SUIT}")
```

♡

Course Files Included

`suits.py`

```
SUITS = '♣ ♦ ♥ ♠'.split()
```

`field()` Specifier

- `default`
- `default_factory`
- `init`
- `repr`
- `compare`
- `hash`
- `metadata`

Position() Example

```
@dataclass
class Position:
    lat: float = 0.0
```

Position() Example

```
@dataclass
class Position:
    lat: float = field(default = 0.0, repr=False)
```

metadata **Parameter**

Next: Representation

Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
3. Basic Data Classes
4. More Flexible Data Classes
 - 4.1 Advanced Default Values
 - ▶ 4.2 **Representation**
 - 4.3 Comparison
5. Immutable Data Classes
6. Inheritance
7. Optimizing Data Classes

Representation

- `repr(obj) - obj.__repr__()`
- `str(obj) - obj.__str__()`

Next: Comparison

Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
3. Basic Data Classes
4. More Flexible Data Classes
 - 4.1 Advanced Default Values
 - 4.2 Representation
 - ▶ 4.3 **Comparison**
5. Immutable Data Classes
6. Inheritance
7. Optimizing Data Classes

Comparison

@dataclass Decorator Parameters

- `init` - Add `__init__()` Method
- `repr` - Add `__repr__()` Method
- `eq` - Add `__eq__()` Method
- `order` - Add Ordering Methods
- `unsafe_hash` - Add `__hash__()` Method
- `frozen` - Assigning to Fields Raises an Exception

Next: Immutable Data Classes

Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
3. Basic Data Classes
4. More Flexible Data Classes
- ▶ 5. **Immutable Data Classes**
6. Inheritance
7. Optimizing Data Classes

Immutable Data Classes

Immutable Data Classes

```
@dataclass(frozen=True)  
class ImmutableDataClass:  
    ...
```

Next: Inheritance

Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
3. Basic Data Classes
4. More Flexible Data Classes
5. Immutable Data Classes
- ▶ 6. **Inheritance**
7. Optimizing Data Classes

Inheritance

Non-Default Arguments after Default Arguments

Non-Default Arguments after Default Arguments

Generated code:

```
def __init__(name: str, lon: float = 0.0, lat: float = 0.0, country: str):  
    ...
```

Non-Default Arguments after Default Arguments

```
@dataclass
class BaseClass:
    value: float = 0.0

@dataclass
class SubClass(BaseClass):
    name: str
```

Invalid Python - Non-Default Argument follows Default from `BaseClass`

Non-Default Arguments after Default Arguments

```
@dataclass
class BaseClass:
    value: float = 0.0

@dataclass
class SubClass(BaseClass):
    name: str = 'Unknown'
```

Valid Python - Only Default Arguments follow Default from BaseClass

SubClass Field Ordering

Next: Optimizing Data Classes

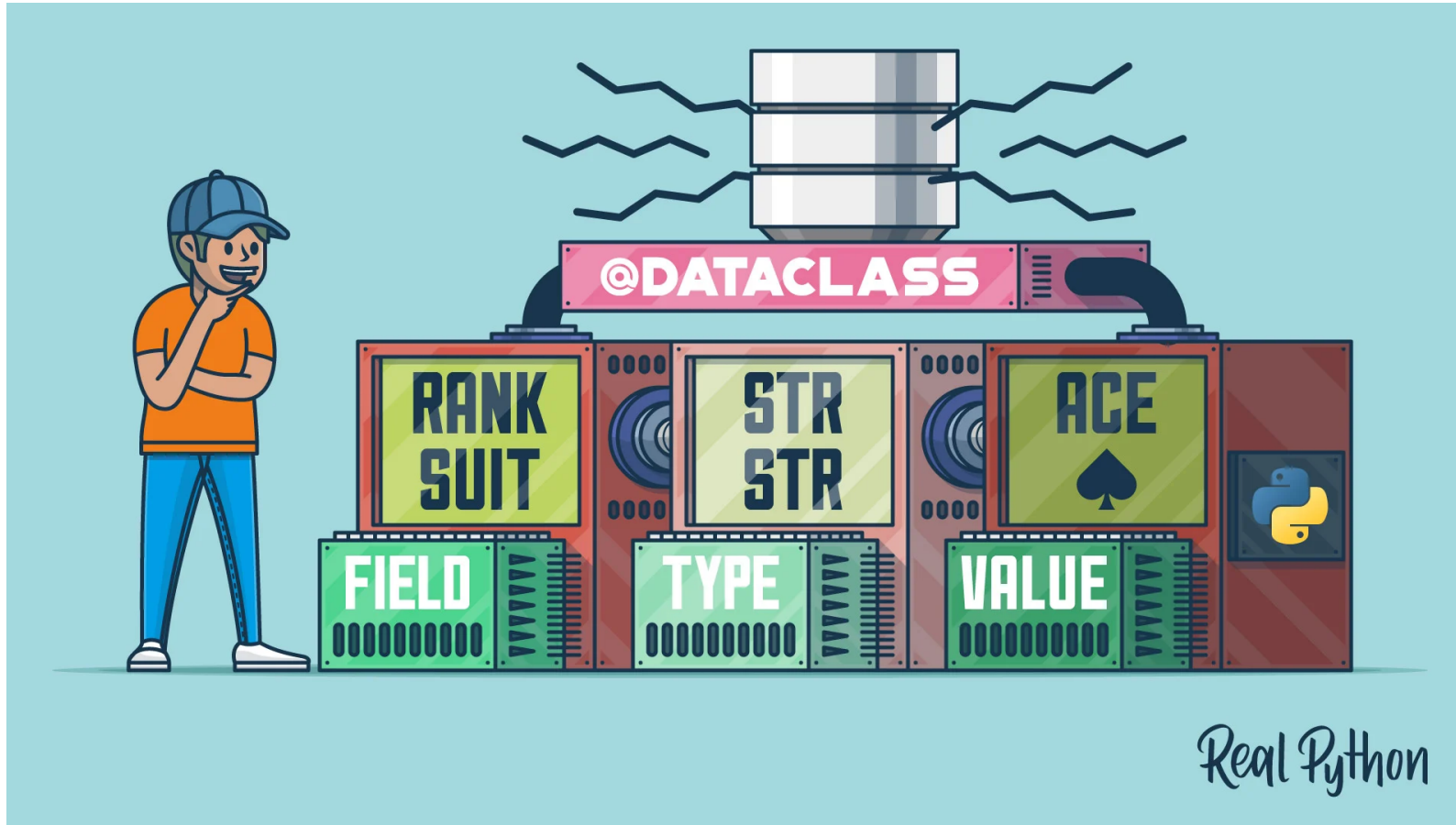
Python Data Classes

1. Comparison to Standard Classes
2. Alternatives to Data Classes
3. Basic Data Classes
4. More Flexible Data Classes
5. Immutable Data Classes
6. Inheritance
- ▶ 7. **Optimizing Data Classes**

Optimizing Data Classes

Next: Summary

Python Data Classes: Summary



Summary

- Defining Data Classes
- Adding Default Values
- Customising Ordering
- Working Immutable Data Classes
- Using Inheritance

Python Data Classes

