circle.py

This script defines a `Circle` class with custom behavior and demonstrates Python's data-model methods.

1. **Class Definition & Constructor**

```
python
CopyEdit
class Circle:
    def __init__(self, radius, name):
        self.__radius = radius  # private attribute
        self.__name = name
```

   a. The `__radius` and `__name` are private due to double underscores.

2. **Getter/Setter for `radius`**

```
python
CopyEdit
@property
def radius(self): ...
@radius.setter
def radius(self, value): ...
```

   a. Enables `c.radius` to get/set, while validating non-negative values.

3. **`get_area` Method**

   Computes area via πr^2 using Python's `math.pi`.

4. **`__str__` & `__repr__`**
   a. `__str__`: user-friendly string
   b. `__repr__`: unambiguous code-style display
5. **`__eq__` & `__hash__`**
   a. `__eq__`: compares either a `Circle` or an `int`.
   b. `__hash__`: uses `radius` so circles with same radius hash identically.
6. **Usage Demo**
   a. Creates `c1`, `c2` with same radius → considered equal & collide in dict keys.
   b. When inserting `c2` in `dict`, it overwrites `c1`.

c. Also shows mixing keys like 5, `'student'`, and hashed `'student'`.

count_num.py

Counts how often numbers appear in a random list.

1. **Generate Random Numbers**
   a. Makes list `numbers` of 50 random integers between 1 and 20.
2. **Print List**
   Shows the full sequence.
3. **Count Occurrences**

```
python
CopyEdit
occur = {}
for num in numbers:
    if num in occur:
        occur[num] += 1
    else:
        occur[num] = 1
```

Builds a dictionary mapping each number to how many times it appears.

employee.py

Defines an `Employee` class and shows how equality & hashing works.

1. **Class Fields**
   `emp_id`, `fname`, `lname`, `salary`.
2. **`__str__` & `__repr__`**
   Both provide nice textual descriptions of the object.
3. **`__eq__`**
   Checks that all fields are identical between two `Employee` instances.

4. **__hash__**
   Creates a hash from a tuple of all fields—necessary when customizing __eq__.
5. **Example Behavior**
   a. Two identical employees produce the same hash.
   b. When used as dict keys, duplicates overwrite previous entries.

📓 enum1.py

Shows how Python's `Enum` class makes code cleaner than using raw strings.

1. **Before `Enum` (commented out)**
   Using plain strings like `"easy"` risks typos and lacks autocomplete.
2. **Defining the `Levels` Enum**

```
python
CopyEdit
class Levels(Enum):
    EASY = 1
    MEDIUM = 2
    HARD = 3
    NIGHTMARE = 4
```

Each member is a constant.

3. **Using Enums**
   a. Comparison: `level == Levels.MEDIUM`.
   b. Function demo: prints a message for HARD.
   c. Access `.value` (e.g. numeric code) and `.name`.
   d. Iterates through all levels.
   e. Shows different ways to access enum members (`Levels(4)` or `Levels['HARD']`).

This is safer and more maintainable than `"medium"` strings.

hash1.py

Explores Python's `hash()` function and data types.

1. **Hashing Example**
    a. Prints `hash(1000)`.
    b. Shows that lists can't be hashed (`hash([1,2])` would fail).
2. **Mutable vs Immutable**
    a. Lists are mutable (changeable) → unhashable.
    b. Tuples, `frozenset`: immutable → hashable.
3. **Custom Class Hash**
    a. Creates `class A`: default hash is based on object identity.
4. **Using Hashes in Collections**
    a. Uses objects/tuples as dict keys and set items, showing which types are allowed.
    b. Prints combined examples like `hash((1, 'hello'))`.

Summary for Beginners

Concept	What You Learned
`@property`	Convert getters/setters into attribute access
`__str__` / `__repr__`	Customize how objects display for humans vs debugging
`__eq__` & `__hash__`	Define equality/hashing so objects behave properly in sets/dicts
`Enum`	Create clear, typo-free collections of constants
Mutable vs Immutable	Know which container types are hashable/unhashable