Python Object-Oriented Programming — Learning Foundation

This guide covers the core OOP topics in Python with concise explanations and runnable code examples. Topics: class, object, constructors and their types, method types, and variable types.

1. Class

A class is a blueprint for creating objects. It defines attributes and methods.

```
class Person:
    """Simple class with attributes and a method"""
    species = "Homo sapiens"  # class variable

    def __init__(self, name, age):
        self.name = name      # instance variable
        self.age = age

    def greet(self):
        return f"Hello, my name is {self.name} and I'm {self.age} years old."

# Example usage
p = Person("Alice", 30)
print(p.greet())  # Hello, my name is Alice and I'm 30 years old.
print(Person.species)  # Homo sapiens
```

2. Object

An object is an instance of a class. Objects hold state and behavior defined by their class.

```
# Creating multiple objects (instances)
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x += dx
        self.y += dy

p1 = Point(1, 2)
p2 = Point(3, 4)
p1.move(2, 3)
print(p1.x, p1.y)  # 3 5
print(p2.x, p2.y)  # 3 4
```

3. Constructor in Python and its types

Constructors initialize new object instances. Types: default (no args), parameterized, and copy-like patterns.

```
# Default constructor (uses default args)
class A:
    def __init__(self, value=0):
        self.value = value

a = A()
print(a.value)  # 0

# Parameterized constructor
```

```
class B:
    def __init__(self, name, score):
        self.name = name
        self.score = score

b = B("Bob", 95)
print(b.name, b.score)  # Bob 95

# Copy-like constructor (create from another instance)
class C:
    def __init__(self, other=None):
        if other is None:
            self.val = 0
        else:
            # copy attributes from other object
            self.val = getattr(other, 'val', 0)

orig = C()
orig.val = 42
copy = C(orig)
print(copy.val)  # 42
```

4. Methods and their types

Methods define behavior. Types: instance methods, class methods, static methods, and special (magic) methods.

```
class Demo:
    counter = 0  # class variable

    def __init__(self, name):
        self.name = name  # instance variable
        Demo.counter += 1

    # Instance method: works on object state
    def greet(self):
        return f"Hi, I'm {self.name}"

    # Class method: receives the class as first arg, used for factory or class-level logic
    @classmethod
    def get_count(cls):
        return cls.counter

    # Static method: utility function with no implicit first arg
    @staticmethod
    def is_adult(age):
        return age >= 18

    # Magic method: control representation
    def __str__(self):
        return f"Demo(name={self.name})"

d = Demo("Cara")
print(d.greet())         # Hi, I'm Cara
print(Demo.get_count())  # 1
print(Demo.is_adult(20)) # True
print(str(d))            # Demo(name=Cara)
```

5. Variables and their types

Variables in classes and methods: instance, class (static), and local variables.

```
class Example:
    class_var = "I am a class variable"  # class/static variable
```

```
    def __init__(self, value):
        self.instance_var = value       # instance variable

    def compute(self, x):
        local_var = x * 2              # local variable
        return local_var

e = Example(10)
print(Example.class_var)    # I am a class variable
print(e.instance_var)       # 10
print(e.compute(5))         # 10
```

Combined Example

```
# Combined example: BankAccount demonstrating OOP concepts
class BankAccount:
    bank_name = "PyBank"  # class variable

    def __init__(self, owner, balance=0):
        self.owner = owner          # instance var
        self.balance = balance      # instance var

    def deposit(self, amount):
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= amount
        return self.balance

    @classmethod
    def info(cls):
        return f"Bank name: {cls.bank_name}"

    @staticmethod
    def interest_rate():
        return 0.02

acct = BankAccount('Dana', 100)
acct.deposit(50)
print(acct.balance)         # 150
print(BankAccount.info())   # Bank name: PyBank
```

Notes

All code examples are concise and ready to run in Python 3. Save this PDF for study and quick reference.