Week 12: Agenda

- Term Project
- OOP + Animations
- Recursion + Animations

OOP: What you need to know

- The idea behind OOP
- Class vs Object
- Functions vs Methods
- How to define a class
- How to use a class
- Inheritance
- Code Tracing

```
1 v class A(object):
          def __init__(self):
 2
              self.b = "sparkling"
 3
                                               32 v def ct3():
 4
                                               33
                                                           b = B("still")
          def f(self):
 5
                                               34
                                                           print(b.h("water"))
 6
              return self.g()
                                               35
                                                           print(type(b) == A)
 7
                                               36
                                                           print(isinstance(b, B))
 8
          def g(self):
                                               37
                                                           print(isinstance(b, A))
              return "apple"
 9
                                               38
                                                           return b
10
                                               39
          def h(self, s):
11
12
              print(f"{self.b} {s}")
                                               40
                                                      print(ct3())
13
              return self.f()
14
          def __repr__(self):
15
16
              return f'A {self.f()}'
17
18
19 V
      class B(A):
20
          def __init__(self, s):
21
22
              super().__init__()
              self.b = s
23
24
25
          def f(self):
26
              s=self.g()
              return f"pine{s}"
27
28
29
          def g(self):
30
              return super().g()
31
```

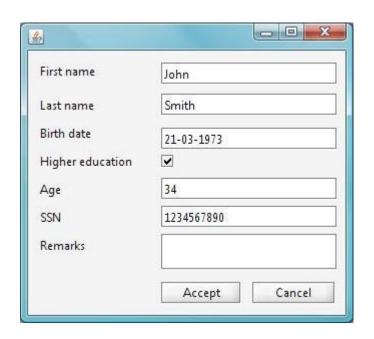
Why GUIs Are Complex

Multiple interacting elements (buttons, windows, events)

Need for state management

User-driven flow (vs. linear scripts)

 Challenge: Keeping things modular, reusable, and testable



OOP is very helpful

- Encapsulation (grouping data + behavior)
- Abstraction (hide details, expose interface)
- Inheritance (reuse GUI components)
- Polymorphism (flexible, dynamic behavior)

GUIs = Hierarchies of objects -> OOP matches this structure

- A Button is an object.
- A Window is an object.
- A EntryField is an object.

A Window has a list of UI elements.

Encapsulation in Action

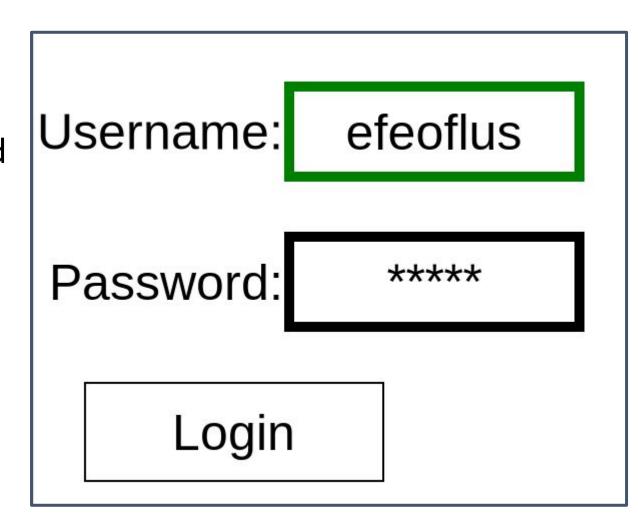
A LoginForm class encapsulates:

- Username field
- Password field
- Submit logic

Changing internal layout doesn't affect other code.

Inheritance for Reuse

- PasswordField extends EntryField
- Override methods: drawContent



Polymorphism for Flexibility

Same **interface**, different behavior:

Common Widget methods on Button, EntryField, ...

draw, isClicked, onClick, onKey,

Event-handling: all widgets implement these methods