Anuncios

1. Recuerden contestar la ECA.
2. Sobre nombres de archivos
3. Hoy: actividad sumativa.
4. EMS.

Encuesta de Medio Semestre

- Su respuesta nos importa.
- Revisaremos sus respuestas y responderemos.
- Pueden acceder mediante Siding.

Les daremos 10 minutos para responderla.

Interfaces gráficas de usuario

Semana 06-07 - Jueves 26 de septiembre de 2019

Interacción con el usuario

- Entradas
- Salidas
- Control de la aplicación

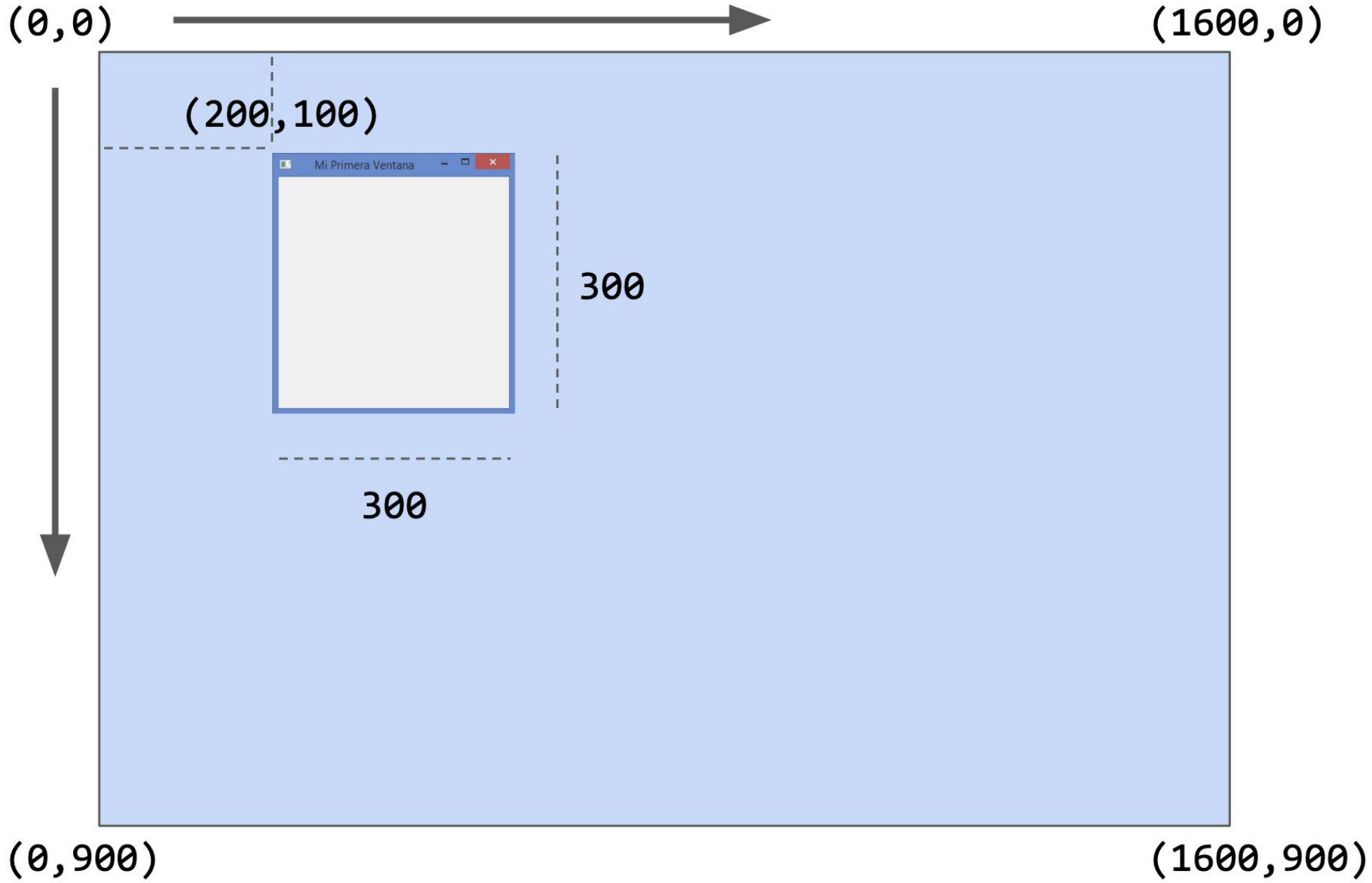
Tipos de interfaces gráficas

- Escritorio
- Móvil
- Web

Tecnologías comunes en Python

- **Escritorio**
  - PyQt
- **Móvil**
  - Kivy
- **Web** ✔✔✔
  - Django
  - Flask

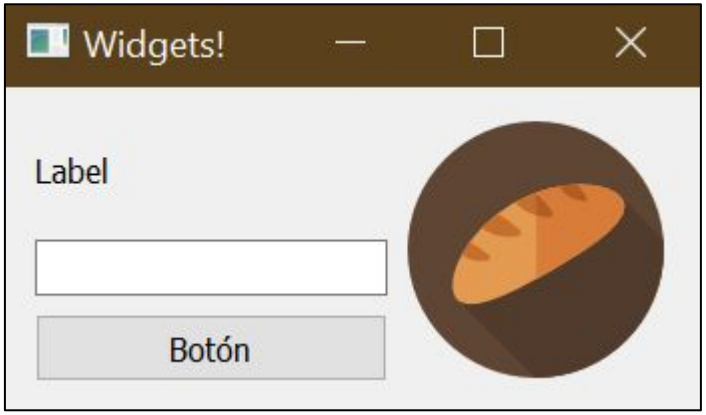
```
class Ventana(QWidget):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.setWindowTitle("Ventana")
        self.setGeometry(200, 200, 400, 400)
        self.show()

app = QApplication([])
ventana = Ventana()
sys.exit(app.exec_())
```


(0,0)
(1600,0)
(200,100)
Mi Primera Ventana
300
300
(0,900)
(1600,900)

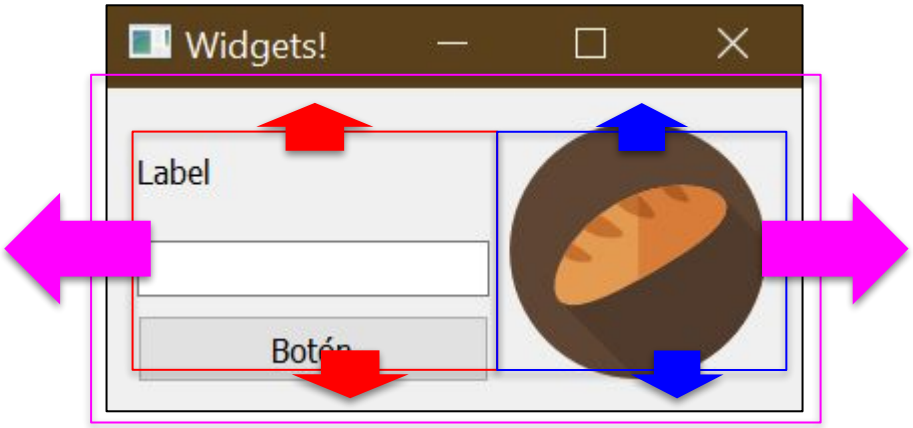
Widgets

- *Labels* y *LineEdits*
- Imágenes
- Botones
- *Layouts*

- *Widgets* personalizados




```
label = QLabel("Label", self)
line_edit = QLineEdit(self)
boton = QPushButton("Botón", self)

imagen = QLabel(self)
imagen.setPixmap(QtGui.QPixmap("ejemplo.png"))
```

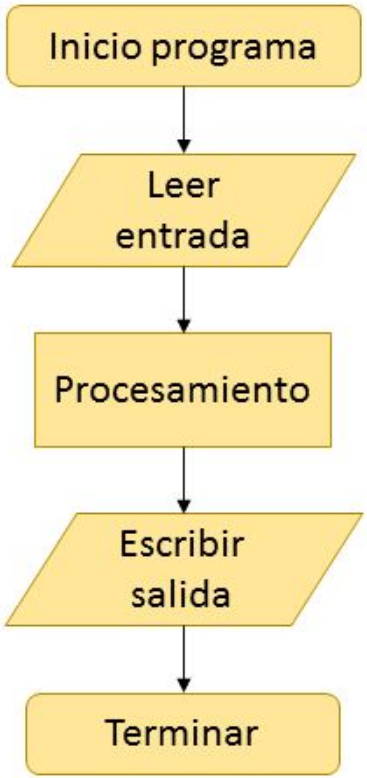



```
layout_izq = QVBoxLayout()
layout_izq.addWidget(label)
layout_izq.addWidget(line_edit)
layout_izq.addWidget(boton)
```

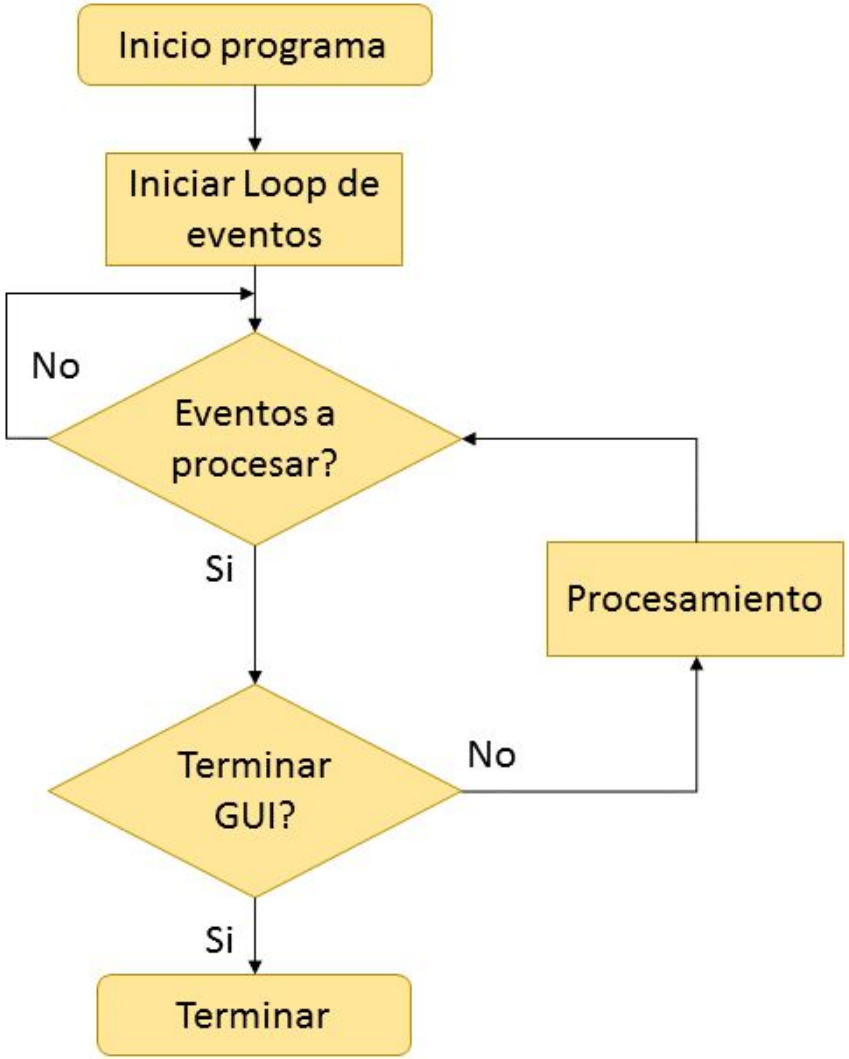
```
layout_der = QVBoxLayout()
layout_der.addWidget(imagen)
```

```
layout_principal = QHBoxLayout()
layout_principal.addLayout(layout_izq)
layout_principal.addLayout(layout_der)
```

```
self.setLayout(layout_principal)
```



a) Programa basado en proceso



b) GUI basada en eventos

Eventos

- `connect(func)`
- `sender()`

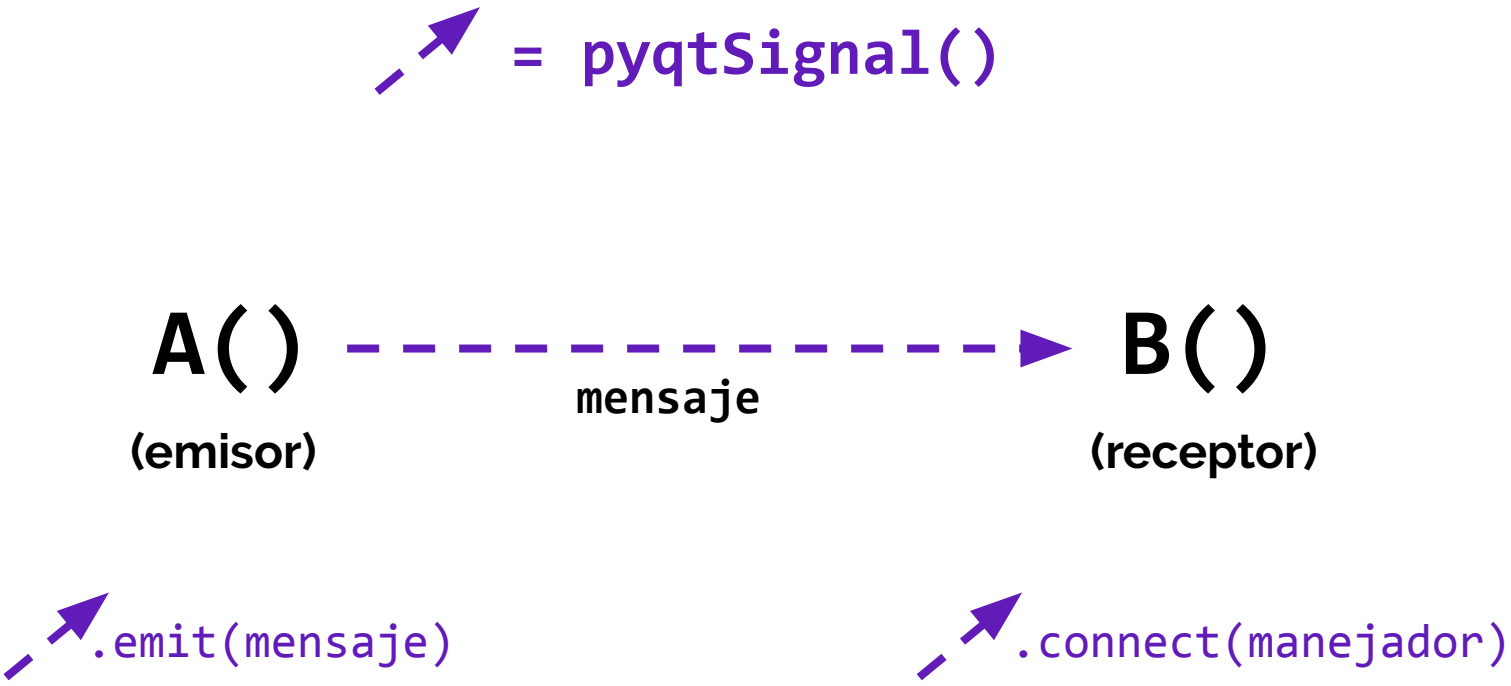
Eventos predefinidos

- `MousePressEvent`
- `KeyPressEvent`

Señales

- emit()
- connect(func)
- emit(mensaje)

Señales


= pyqtSignal()
A()
(emisor)
mensaje
B()
(receptor)
.emit(mensaje)
.connect(manejador)

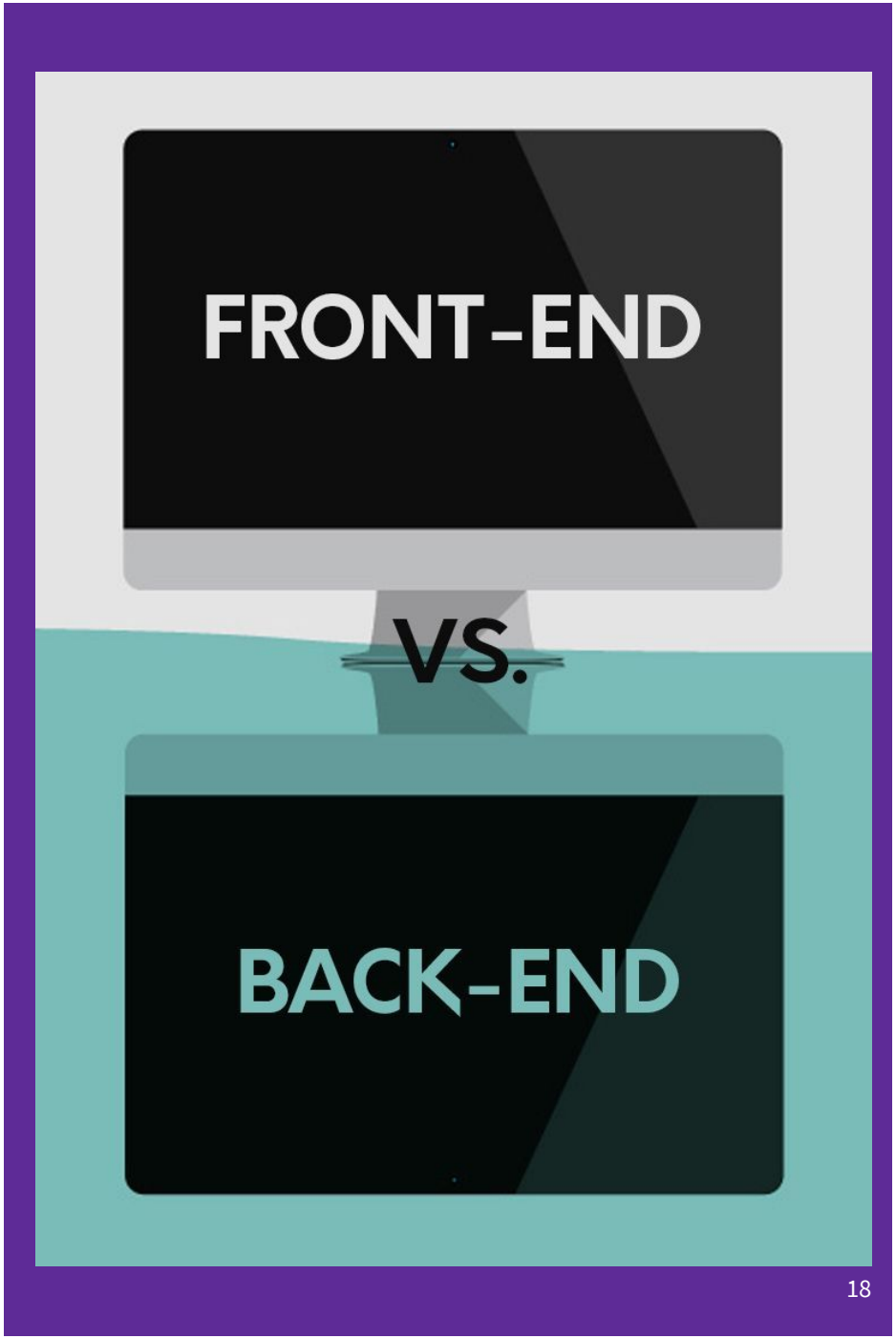
Veamos código

(Hora de abrir el editor)

En este ejemplo veremos:

- `clicked.connect`
- *Front-end* y *back-end*
- Uso esperado de señales

Patrón de diseño *Front-end* *Back-end*



Patrón de diseño *Front-end* *Back-end*

BAJO acoplamiento y ALTA cohesión

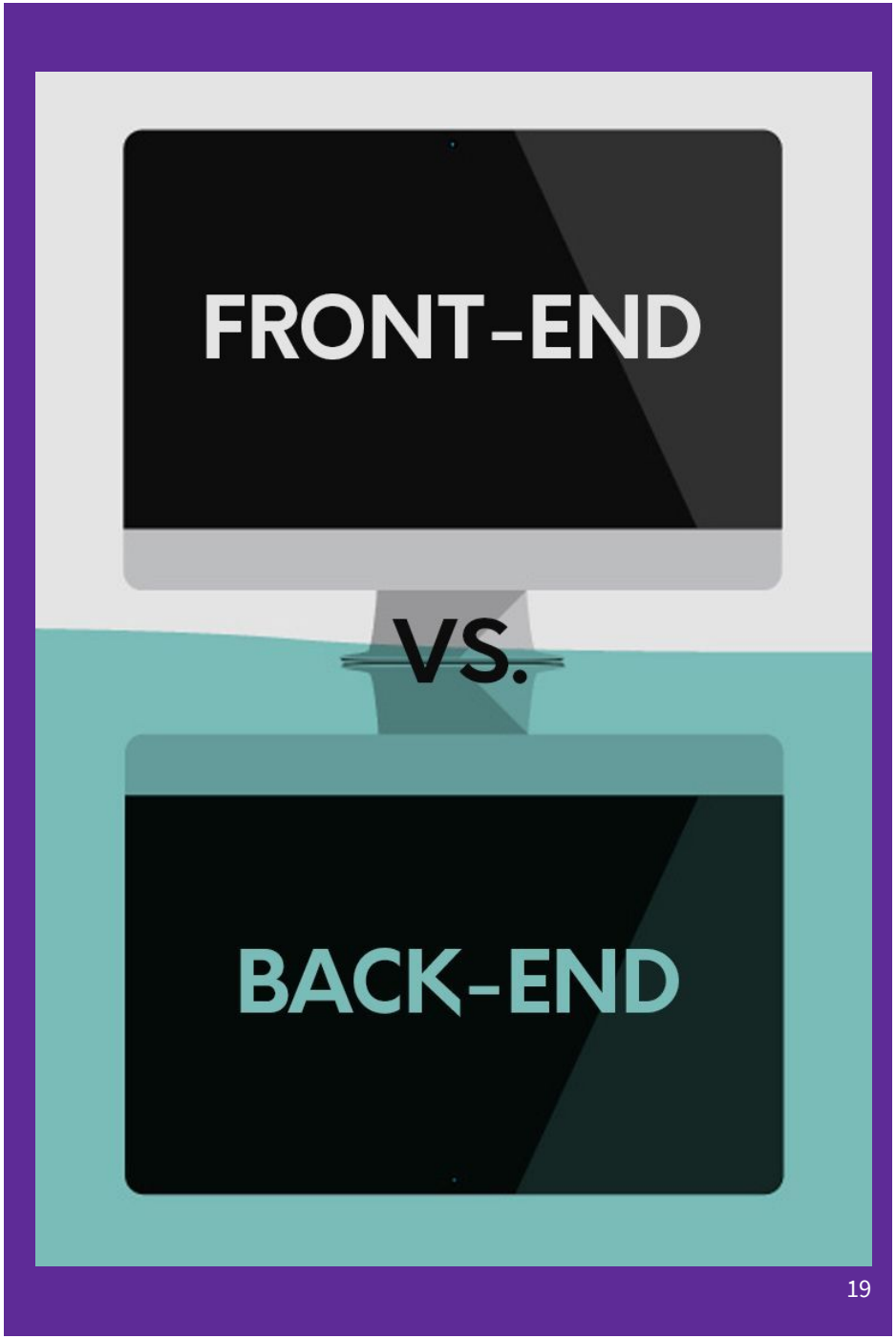
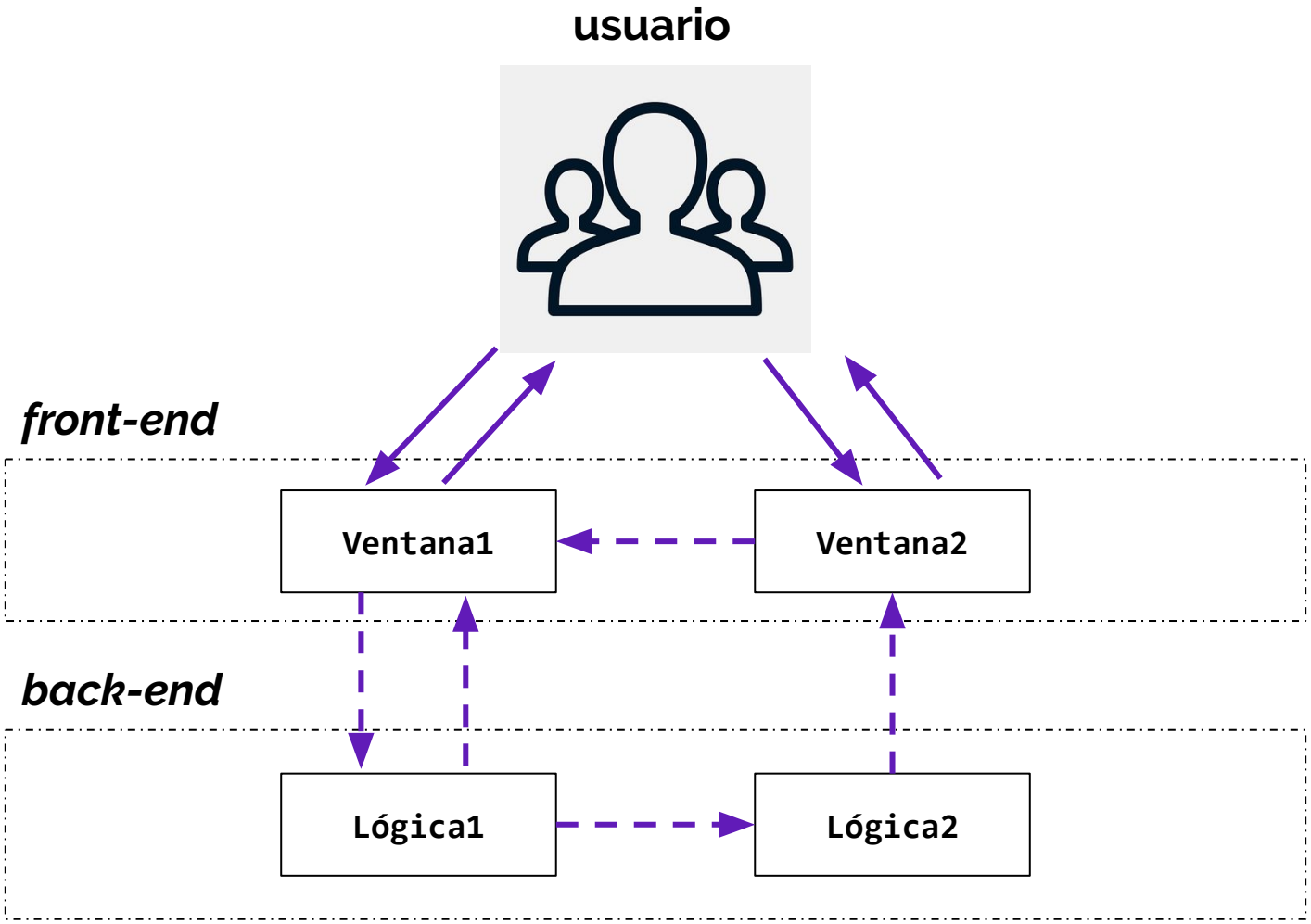


Diagrama de modelación



Dependencias circulares ❌❌❌

```python
# front-end
class Ventana(QWidget):

    def __init__(self):
        self.logica = Logica(self)
        ...

    def enviar(self, msj):
        ...
        self.logica.enviar(msj)
        ...
```

```python
# back-end
class Logica:

    def __init__(self, ventana):
        self.ventana = ventana
        ...

    def actualizar_ventana(self):
        ...
        self.ventana.actualizar()
        ...
```

`self.logica.ventana == self`

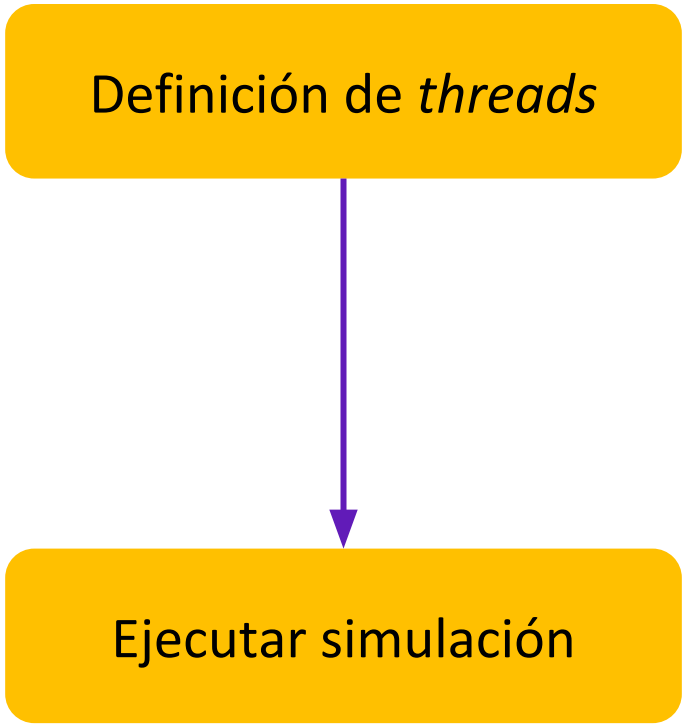
`self.ventana.logica == self`

Actividad

1. En el *syllabus*, vayan a la carpeta “Actividades” y descarguen el enunciado de la actividad 05 (AC05). https://github.com/IIC2233/syllabus
2. Trabajen **de forma individual** hasta las **16:45**.
3. Recuerden hacer *commit* y *push* cada cierto tiempo.

Cierre

Diagrama de flujo de AC05




```
class Excavador(Thread)
class Imprimidor(Thread)
```

```
def comenzar()
def estadisticas()
```

Threading

¿Se podría haber hecho la AC sin el uso de *threads*?

Threads proveen una interfaz y forma única de modelar situaciones de ejecución concurrente.

Simular este comportamiento con programas *single-threaded* es mucho más difícil.

Threads

```
# main.py
...
for excavador in self.excavadores:
    excavador.start()
for imprimidor in self.imprimidores:
    imprimidor.start()
...
```

```
# excavadores.py
class Excavador(Thread):
    ...

    def run(self):
        # comportamiento
```

```
# imprimidores.py
class Imprimidor(Thread):
    ...

    def run(self):
        # comportamiento
```

Threads

```
# excavadores.py
class Excavador(Thread):

    def __init__(self, nombre, berlin, tunel):
        super().__init__() # importante
        self.nombre = nombre
        self.berlin = berlin
        self.tunel = tunel

    def run(self):
        while self.tunel.metros_avanzados < self.tunel.largo:
            reloj(10) # demora 10 minutos
            self.avanzar(randint(50, 100))
            if uniform(0, 1) <= 0.1:
                self.problema_picota()
```

Próxima semana

1. Se publica este sábado la Tarea 2.
2. Actividad formativa de semana 8: Excepciones.