```python
def repaso_examen():
    print('Introducción a la Programación')
```

Strings

```
nuevo_string = 'Se pasa Intro a la Progra?'
otro_string = "Obvio que si!!"
```

```
string_largo = ('Esto es un string bastante largo. Por alguna razón, lo estoy '
                'escribiendo a las 3 A.M.\nAcabo de saltarme una línea, pero no '
                'sé por qué lo hice.\nSólo lo hice.\nMe pregunto cuántos de '
                'ustedes estarán despiertos a esta hora.')
```

Output: Esto es un string bastante largo. Por alguna razón, lo estoy escribiendo a las 3 A.M.
Acabo de saltarme una línea, pero no sé por qué lo hice.
Sólo lo hice.
Me pregunto cuántos de ustedes estarán despiertos a esta hora.

```
nuevo_string = 'Se pasa la analogía de Intro a la Progra?'
nuevo_string[0]  # 'S'
nuevo_string[3]  # 'p'
nuevo_string[-1]  # '?'
nuevo_string[-3]  # 'r'
nuevo_string[11:14]  # 'ana'
```

```python
nuevo_string = 'HOLA GENTE'
for letra in nuevo_string:
  print(letra)
```

Output:

```
'H'
'O'
'L'
'A'
' '
'G'
'E'
'N'
'T'
'E'
```

```python
nuevo_string = 'FLIPITY-FLAPITY-FLUPITY'
lista_rara = nuevo_string.split('-')
print(lista_rara)  # ['FLIPITY', 'FLAPITY', 'FLUPITY']
```

Listas

```python
una_lista_cualquiera = ['una', 'lista', 'cualquiera']  # creatividad nivel 9999
```

```
otra_lista_cualquiera = [['una', 'lista', 'cualquiera'],
                         ['dos', 'listas', 'cualquiera'],
                         ['suerte', 'en el', 'examen']]
```

```
.append(elemento)
.remove(elemento)
.pop(indice)
.insert(indice, elemento)
```

```python
una_lista_cualquiera = ['una', 'lista', 'cualquiera']
una_lista_cualquiera[0]  # 'una'
una_lista_cualquiera[-1]  # 'cualquiera'
una_lista_cualquiera[1]  # 'lista'
una_lista_cualquiera[:-1]  # ['una', 'lista']
una_lista_cualquiera[0:2]  # ['una', 'lista']
```

Programación Orientada a Objetos

```python
class Persona:
    pass
```

¿Qué quiere decir esto? Quiere decir que existen objetos que se pueden agrupar debido a que tienen características similares. A este grupo de objetos los llamamos 'Persona'. Lo que se defina dentro de 'Persona' será código que todas las instancias de persona compartirán.

```python
class Persona:

    def __init__(self, nombre_externo):
        self.nombre = nombre_externo
        self.especie = 'Humano'
```

Hemos creado el 'constructor' de la clase. Este método se llamará cada vez que se instancie la clase. Lo único que hace este método es definir que cualquier instancia de 'Persona' será de especie 'Humano' y tendrá como nombre el primer parámetro entregado al constructor de la clase (más de esto en la siguiente diapositiva).

```python
class Persona:

    def __init__(self, nombre_externo):
        self.nombre = nombre_externo
        self.especie = 'Humano'


persona_1 = Persona('Daniel')
```

Hemos instanciado una clase! ¿Qué quiere decir esto? Quiere decir que ahora existe un objeto que guardamos con el nombre de 'persona_1' que puede ser clasificado como 'Persona'. El primer argumento que se le dio al constructor de clase fue 'Daniel', por lo que persona_1 tiene como nombre 'Daniel'. Además, a todas las personas se les asigna automáticamente su especie, ya que todas las personas son Humanos (¿o no?).

```python
class Persona:

    def __init__(self, nombre_externo):
        self.nombre = nombre_externo
        self.especie = 'Humano'


persona_1 = Persona('Daniel')
persona_2 = Persona('Bryan')
```

Al igual que en la diapositiva anterior, hemos instanciado a la clase Persona. Esta vez, hay dos personas distintas. La primera tiene como nombre 'Daniel', y la segunda tiene como nombre 'Bryan'. Como ambas son instancias de Persona, la especie de ambas se setea automáticamente a 'Humano'.

Los objetos también pueden tener atributos que sean otros objetos!

```python
class GrupoAmigos:

    def __init__(self, nombre_grupo, lista_amigos):
        self.nombre = nombre_grupo
        self.amigos = lista_amigos
```

Acabamos de definir una clase que agrupará cualquier tipo de grupo de amigos. El constructor de esta clase recibe el nombre del grupo de amigos como primer parámetro y una lista de amigos como segundo parámetro.

```python
class GrupoAmigos:
    def __init__(self, nombre_grupo, lista_amigos):
        self.nombre = nombre_grupo
        self.amigos = lista_amigos

    def mostrar_nombres_amigos(self):
        for amigo in self.amigos:
            print(amigo.nombre)

class Persona:
    def __init__(self, nombre_externo):
        self.nombre = nombre_externo
        self.especie = 'Humano'

persona_1 = Persona('Daniel')
persona_2 = Persona('Bryan')
persona_3 = Persona('Gonzalo')
persona_4 = Persona('Pelao')
amigos = [persona_1, persona_2, persona_3, persona_4]

grupo_amigos = GrupoAmigos('Las Rocas', amigos)
```

Con este trozo de código definimos una clase que agrupará grupos de amigos y una clase que agrupará personas. Además, instanciamos 4 personas con distintos nombres y luego instanciamos un grupo de amigos ficticio llamado 'Las Rocas' que contiene a las 4 personas previamente instanciadas. Sin embargo, hay un método que no fue usado. Este método imprime en consola los nombres de todos los amigos de un grupo de amigos. Como no fue llamado, no se imprimirán en consola los nombres de los amigos del grupo 'Las Rocas'.

```python
class GrupoAmigos:
    def __init__(self, nombre_grupo, lista_amigos):
        self.nombre = nombre_grupo
        self.amigos = lista_amigos

    def mostrar_nombres_amigos(self):
        for amigo in self.amigos:
            print(amigo.nombre)

class Persona:
    def __init__(self, nombre_externo):
        self.nombre = nombre_externo
        self.especie = 'Humano'

persona_1 = Persona('Daniel')
persona_2 = Persona('Bryan')
persona_3 = Persona('Gonzalo')
persona_4 = Persona('Pelao')
amigos = [persona_1, persona_2, persona_3, persona_4]

grupo_amigos = GrupoAmigos('Las Rocas', amigos)
grupo_amigos.mostrar_nombres_amigos()
```

Con este trozo de código si se llama al método para ver los nombres de los amigos. El output será algo parecido a lo siguiente:

```
Daniel
Bryan
Gonzalo
Pelao
```

```python
class Persona:

    def __init__(self, nombre_externo):
        self.nombre = nombre_externo
        self.especie = 'Humano'
        self.apodo = None

    def cambiar_apodo(self, nuevo_apodo):
        self.apodo = nuevo_apodo

persona_1 = Persona('Bryan')
persona_2 = Persona('John Cena')
persona_1.cambiar_apodo('Guatón Byron')
persona_2.cambiar_apodo('Tinky Winky')
```

Hemos cambiado un poco la clase Persona. Ahora, las personas pueden tener apodos! Por supuesto que al nacer nadie tiene un apodo (creo), pero a lo largo de la vida una persona podría adquirir un apodo! En el ejemplo, Bryan nace sin apodo, pero le es asignado uno, al igual que a John Cena.

Lo anterior también se podría hacer sin tener un método específico para ello!!

```
class Persona:

    def __init__(self, nombre_externo):
        self.nombre = nombre_externo
        self.especie = 'Humano'
        self.apodo = None


persona_1 = Persona('Bryan')
persona_2 = Persona('John Cena')
persona_1.apodo = 'Guatón Byron'
persona_2.apodo = 'Tinky Winky'
```

Recursión y BackTracking

```python
def recursion():
    recursion()
```

Esta función nunca terminará!!

```python
def recursion(numero):
    if numero <= 0:
        return
    print(numero)
    recursion(numero - 1)

recursion(3)
```

Qué pasará?

```
3
2
1
```

Idea General

Queremos llamar la función con algunos argumentos y chequear si tenemos lo necesario para detener la función (caso base). Si ya tenemos lo que queremos, retornamos algo. En otro caso, ejecutamos la función de nuevo (generalmente se alteran levemente los argumentos) y se vuelve a llamar a la función con los argumentos levemente alterados.