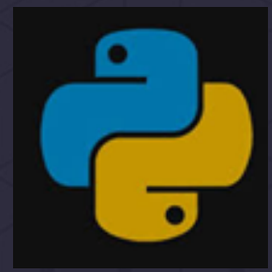
PREGRADO

UNIDAD 1 | SEMANA 2

ESTRUCTURAS DE CONTROL, FUNCIONES, STRING, BUCLES



1ACC0201| Programación Orientada a Objetos

Al finalizar de la sesión el alumno entiende las estructuras selectivas, las funciones de usuario, el manejo de string y el uso de for para recorrer iterables

AGENDA

1. Estructura de Control
   - operadores
   - if-elif-else-match
   - La indentación
2. Funciones de usuario
3. strings
   - Objetos de clase string
   - Índices, slicing
   - Función len()
   - Bucles for: iterando strings
   - F-string

Operadores de comparación y lógicos

De comparación

OPERADOR	NOMBRE	EJEMPLO
==	Igualdad	A == B
!=	Diferente	A != B
>	Mayor	A > B
<	Menor	A < B
>=	Mayor o igual	A >= B
<=	Menor o igual	A <= B

Lógicos

OPERADOR	DESCRIPCION	EJEMPLO
and	Retorna verdadero si ambas condiciones son verdaderas	A and B
or	Retorna verdadero si una de las condiciones es verdadera	A or B
not	Niega el resultado de la condición	not (A)

Operador is

OPERADOR	DESCRIPCION	EJEMPLO
is	Comprueba si dos variables apuntan al mismo objeto en la memoria, si es así retorna verdadero	A is B
is not	Retorna verdadero si las variables no apuntan al mismo objeto	A is not B

Estructuras de Control Selectivas

- Sentencia **if-else**

```
if numero > 0:
    print("Es positivo")
else:
    print("No es positivo")
```

- Al final de la condición siempre van dos puntos
- En Python no hay llaves, lo que **indica** cuales es el **bloque de sentencias** si la **condición es verdadera** es la **indentación**.
- La indentación, según las normas de buena codificación Python (PEP8) debe ser de **4 espacios**

Estructuras de Control Selectivas

```
temperatura = 28
if temperatura < 20:
    if temperatura < 10:
        print("Nivel azul")
    else:
        print("Nivel verde")
else:
    if temperatura < 30:
        print("Nivel naranja")
    else:
        print("Nivel rojo")
```

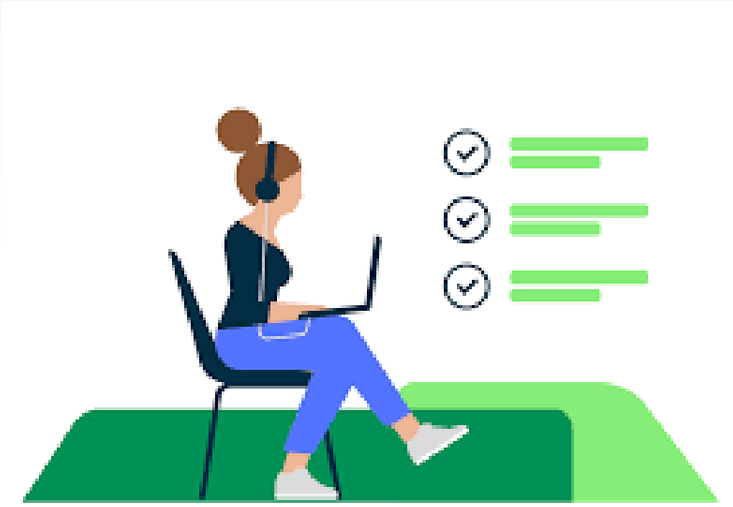
Nivel naranja

```
temperatura = 34
if temperatura < 10:
    print("Nivel azul")
elif temperatura < 20:
    print("Nivel verde")
elif temperatura < 30:
    print("Nivel naranja")
else:
    print("Nivel rojo")
```

Nivel rojo

Estructuras de Control Selectivas: match

- En C++ existe una declaración condicional conocida como Switch Case.
- **Match Case** es el equivalente en Python, que se introdujo desde la versión Python 3.10.
- Aquí primero tenemos que pasar un parámetro y luego intentar comprobar con qué caso se satisface el parámetro.
- Si encontramos una coincidencia, ejecutaremos algún código y, si no hay ninguna coincidencia, se llevará a cabo una acción predeterminada.



Estructuras de Control Selectivas: match

```python
# Declaración de caso de match simple
def runMatch():
    num = int(input("Introduzca un número entre 1 y 3: "))
    match num:
        case 1: print("Uno")
        case 2: print("Dos")
        case 3: print("Tres")
        case _: print("Número no está entre 1 y 3")


# Programa principal
runMatch()
```

```python
comando = "¡Hola mundo!"
match comando:
    case "¡Hola mundo!":
        print("¡Hola a ti también!")
    case "¡Adiós, mundo!":
        print("Nos vemos luego")
    case other:
        print("No se encontró ninguna coincidencia")
```

Estructuras repetitivas: while

- Se basa en repetir un bloque a partir de evaluar una condición lógica, siempre que ésta sea True.
- Queda en las manos del programador decidir el momento en que la condición cambie a False para hacer que el while finalice.

```
n = 5
while n > 0:
    print(n)
    n -= 1
print('¡Despegue!')
print(n)
```

Se verá el siguiente resultado

```
5
4
3
2
1
¡Despegue!
0
```

while: usando break

Break se usa para terminar un bucle

Escribir y ejecutar el siguiente ejemplo y analizar los resultados

```
while True:
    nota = float(input("Ingresar nota entre 0 y 20 "))
    if 0 <= nota <= 20:
        break
    else:
        print("Nota fuera de rango válido")
print("La nota ingresada fue", nota)
```

while: usando break

Se puede reforzar la validación de datos usando try y except

```
while True:
    try:
        nota = float(input("Ingresar nota entre 0 y 20 "))
        if 0 <= nota <= 20:
            break
        else:
            print("Nota fuera de rango válido")
    except ValueError:
        print("Nota ingresada no es un número")

print("La nota ingresada fue", nota)
```

```
Ingresar nota entre 0 y 20  ww
Nota ingresada no es un número
Ingresar nota entre 0 y 20  25
Nota fuera de rango válido
Ingresar nota entre 0 y 20  -5
Nota fuera de rango válido
Ingresar nota entre 0 y 20  18
La nota ingresada fue 18.0
```

while: usando continue

Sirve para "saltarse" la iteración actual sin romper el bucle.

```
c = 0
while c <= 5:
    c += 1
    if c == 3 or c == 4:
        print("Continuamos con la siguiente iteración", c)
        continue

    print("c vale", c)

print("Se ha completado toda la iteración y c vale", c)
```

```
c vale 1
c vale 2
Continuamos con la siguiente iteración 3
Continuamos con la siguiente iteración 4
c vale 5
c vale 6
Se ha completado toda la iteración y c vale 6
```

La importancia de la indentación en Python

Una de las características más distintivas de Python es el uso de indentaciones para marcar bloques de código.

```
if pwd == 'manzana':
    print('Iniciando sesión ...')
else:
    print('Contraseña incorrecta.')

print('¡Todo terminado!')
```

- Las líneas print('Iniciando sesión ...') y print('Contraseña incorrecta') son dos bloques de código separados.
- Para indicar un bloque de código en Python, debes indentar cada línea del bloque en la misma cantidad.
- Los dos bloques de código en nuestro ejemplo sentencia if están indentados con cuatro espacios que es una cantidad típica de sangría para Python.
- En la mayoría de los otros lenguajes de programación, la indentación se usa solo para ayudar a que el código se vea bonito. Pero en Python, es necesario para indicar a qué bloque de código pertenece una declaración. Por ejemplo, la sentencia final print ("¡Todo terminado!") No está endentada, por lo que no es parte del bloque else.

Funciones definidas por el usuario

- ¿Qué es una función definida por el usuario?
  - Una **función** es un bloque de código que **solo** se ejecuta cuando se llama o invoca.
  - Se puede pasar datos, conocidos como parámetros o argumentos, a una función.
  - Una función puede devolver datos como resultado o no devolver nada.
  - Comienza con la palabra **def**
- Cualquier función tendrá:
  - Un nombre
  - Argumentos de entrada (opcional)
  - Un código para ejecutar
  - Unos parámetros de salida (opcional)

Funciones definidas por el usuario

- Veamos un ejemplo

```
def funcion(x):
    return x**2


# Programa principal
y = funcion(3)
print(y)
```

```
9
```

- En Python una función puede retornar más de un valor

```
def mifuncion(x):
    return x * 2, x**2


# Programa principal
doble, cuadrado = mifuncion(5)
print(doble, cuadrado)
```

```
10 25
```

Funciones definidas por el usuario

- Usando más de un parámetro

```
def resta(a, b):
    return a - b


# Programa principal
print(resta(2, 4))
print(resta(4, 2))
```

```
-2
2
```

- Parámetros por defecto

```
def suma(a, b, c=0):
    return a + b + c


# Programa principal
print(suma(1, 2, 3))
print(suma(5, 6))
```

```
6
11
```

- Las funciones pueden no retornar nada

```
def suma(a, b, c=0):
    print(a + b + c)


# Programa principal
suma(1, 2, 3)
suma(5, 6)
```

```
6
11
```

Funciones definidas por el usuario

```
def diaNombre(n):
    match n:
        case 0: return "Lunes"
        case 1: return "Martes"
        case 2: return "Miércoles"
        case 3: return "Jueves"
        case 4: return "Viernes"
        case 5: return "Sábado"
        case 6: return "Domingo"
        case _: return "Número de día no válido"


# pp
print(diaNombre(3))
print(diaNombre(6))
print(diaNombre(7))
```

Đº ửğŝǀ ửì źĂ ì ļ ửŝ ẻźĄ ǀ ỷĂ ụĂº źº ẻỷŝǀ

- **Ọẻỷŝẻğụ źĂ ǀ ğẻụ źĂ Ă ǀ ğẻụ ź**
  - ĻŝĂì Ăẻźŝ ửẻĂ Ă ì ļ ửì Ăº ửẻĂẺẻỷŝẻğụ Ă ửĂ ụĂ ỷǀ ɔỷẻừ ẻĂ ỷŝửğŝ ẻụĞửǀ Ă ửĂẻĂ ķº ửğŝǁử7Ăì Ăẻźº ử ŝỷẹĂº ì Ă źĂ ụǀ ğẻụ9
  - Ļì Ăº ì î ì Ăğỷì ẻỷĂº ửẻĂẺẻỷŝẻğụ Ă ì ụĂử źử ǀĂǀ ử ğỷì Ă ì ửởỷǀ Ă ì Ăẻķº ửğŝǁử9
  - Fẻ Ăº ì Ăº ì Ăğỷì ẻî ẻĂ ì ửởỷǀ Ă ì Ăẻķº ửğŝǁửĂì ỷẹĂ ǀ ğẻụĂ ǀ ử ẻỷẹĂ ŝğř ǀĂẻụ ỷĂ ửĂ ụẻĂķº ửğŝǁử9
  - FẻĂẺẻỷŝẻğụ Ă ụǀ ğẻụĂử ẻửĉ ửî ỷẹĂ ụẺẻụ ỷĂŝửğŝẻụ9
  - Cźĉ ẻĂẺẻỷŝẻğụ Ăº ì î ì Ăì ỷĂº źẻî ẻĂ ǀ ỷĂẻĂķº ửğŝǁửĞĂì ỷǀ Ăǀ Ăử ǀî ŝ ğẻî ẻĂ ửĂº Ă Ẻẻụ ỷ
  - Ň ỷử ŝửẻî ẻĂẻĂ ţì ğº ğŝǁửĂ ì Ăẻķº ửğŝǁửĞẻĂẺẻỷŝẻğụ Ă ǀ ğẻụĂ ì źẻỷẻỷì ğì ỷẹĞĂĂ ỷº ì î ẻỷẹĂẻĂ ụǀ ğẻụĞǀ ửĂº ĂẺẻụ ỷĂ ỷŝửẻụ9

Funciones definidas por el usuario

- Veamos un ejemplo

```
def resta(a, b):
    x = 10
    print(x, id(x))
    return a - b


# Programa principal
a = 2
b = 4
x = 5

print(resta(a, b))
print(x, id(x))
```

Pregunta para análisis:
¿Qué significan esos resultados?

```
10 140731795446488
-2
5 140731795446328
```

Funciones definidas por el usuario

- Veamos un ejemplo

```
def resta(a, b):
    print(x, id(x))
    return a - b


# Programa principal
a = 2
b = 4
x = 5

print(resta(a, b))
print(x, id(x))
```

Pregunta de desafío:
¿Cómo se explican los resultados?

```
5 140731795446328
8
5 140731795446328
```

string

Cadenas en Python: string

- Los strings de Python son uno de los tipos de datos más importantes del lenguaje.
- Se escriben entre comillas simples o dobles y se muestran mediante la función print.
  - `cadena1 = “Hola”`
  - `cadena2 = ‘Hola’`
- Un string en Python es básicamente una secuencia de varios caracteres individuales
- Las cadenas no se pueden modificar
- En Python se puede escribir textos largos, de varias líneas, encerrándolos en triples comillas, sean estas simples o dobles. Veamos el siguiente ejemplo:

```
texto = """
el payaso corrió detrás del coche y el coche se metió en la tienda y
la tienda cayó sobre el payaso y el coche
"""
```

string

Cuando se crea un string, Python le crea índices
Supongamos hacemos:
holaMundo = “Hola, mundo.”
Python le crea índices al string, tal como se muestra en la figura siguiente

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Se crean índices positivos y negativos, numerándose desde cero de izquierda a derecha y desde -1 de derecha a izquierda

string

Se puede acceder a caracteres individuales de un string a través de sus índices, encerrando el valor del índice entre corchetes []

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
holaMundo = "Hola, mundo."
print(holaMundo[4])
```

Resultado

```
,
```

No se puede cambiar un valor individual de una cadena, la siguiente sentencia producirá un error:

```
holamundo[4] = “.”
```

string: slicing

Se puede acceder a un grupo de caracteres de un string a través del uso del slicing, cuya sintaxis es la siguiente: **cadena[inicio:final:paso]**

- Si el "paso" es positivo, se lee de izquierda a derecha
- Si el "paso" es negativo, se lee de derecha a izquierda
- Si no se pone "paso", el valor por defecto es 1 positivo
- Si no se pone "inicio", se entiende que es desde el primero elemento del string
- Si no se pone "final", se entiende que es hasta el último elemento del string si el paso es positivo o el inicio si el paso es negativo
- No se incluye a carácter cuyo índice es el que figura como "final"

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
holaMundo = "Hola, mundo."
print(holaMundo[:4])
```

Resultado

Hola

string: slicing

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a	,		m	u	n	d	o	.
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Ejecute y analice los resultados del siguiente código

```
holaMundo = "Hola, mundo."
print(holaMundo[:4:2])
print(holaMundo[-6:])
print(holaMundo[-6:-1])
print(holaMundo[-8::1])
print(holaMundo[-8::-1])
print(holaMundo[::-1])
```

Resultados

```
Hl
mundo.
mundo
, mundo.
,aloH
.odnum ,aloH
```

string: operadores + y *

En los string el signo ‘+’ significa concatenación

```
nombre = "Pablo"
apellido = "Valdivia"
nombreCompleto = nombre + " " +  apellido
print(nombreCompleto)
```

Pablo Valdivia

En los string el signo ‘*’ seguido de un número se usa para repetir strings

```
# operador * en strings
# repetir strings
cadena = "hola " * 4
print(cadena)
```

hola hola hola hola

string: comparación

Se pueden comparar cadenas, Python compara carácter por carácter y se hace a través de la codificación ASCII

```
resultado1 = "hola" == "hOla"
resultado2 = "hola" > "hOla"

print(resultado1)
print(resultado2)
```

```
False
True
```

La función len()

- Devuelve el número de caracteres del string

```
cadena = "Hola mundo"
print(len(cadena))
```

```
10
```

Bucles “for” en Python

- Los bucles **for** en Python son un tipo especial de sentencia que se utiliza para el recorrido secuencial.
- El bucle **for** de Python se utiliza para iterar sobre un elemento iterable como un string, una tupla, una lista, o un diccionario.
- En Python, no existe un bucle **for** al estilo C++, es decir, for (i=0; I <n; i++)
- La sintaxis básica es la siguiente:

```
for variable in iterable:
    # código python
```

- Hay dos palabras clave, **for** e **in**
- Observe también la indentación que señala el bloque de código que pertenece al **for.** Veamos un ejemplo recorriendo un string

```
holaMundo = "Hola, mundo."
for caracter in holaMundo:
    print(caracter)
```

Ejecutar y analizar los resultados

enumerate()

- En Python, un bucle **for** generalmente se escribe como un bucle sobre un objeto iterable.
- Esto significa que no necesita una variable de recuento para acceder a los elementos del iterable.
- A veces, sin embargo, desea tener una variable que cambie en cada iteración del bucle.
- En lugar de crear e incrementar una variable, se puede usar **enumerate()** de Python para obtener un **contador** y el **valor** del iterable al mismo tiempo.
- La sintaxis es la siguiente:

```
enumerate(iterable, start=0)
```

Donde:

- Iterable: cualquier objeto que admita iteración.
- start: el valor del índice desde el cual se iniciará el contador, por defecto es 0

```
cadena = "Hola mundo"

for i, valor in enumerate(cadena):
    print(i, valor)

for i, valor in enumerate(cadena, 1):
    print(i, valor)
```

Ejecutar y analizar los resultados

Formateando cadenas: f-strings

- También llamados "literales de cadena con formato", las **cadenas f** son literales de cadena que tienen un **f al principio** y **llaves** que contienen expresiones que se reemplazarán con sus valores en tiempo de ejecución.
- Se llaman cadenas 'f' porque es necesario prefijar una cadena con la letra 'f' para crear una cadena f-string. La letra 'f' también indica que estas cadenas se utilizan para formatear. Los f-string son la forma más simple y práctica de formatear cadenas.
- Un ejemplo de cómo se usa el f-string es el siguiente:

```
nombre, apellido = "Jhon", "Lennon"
print(f"Buenas tardes Sr. {nombre} {apellido}, esperemos se encuentre bien")
print(f"Buenas tardes Sr. {apellido} {nombre}, esperemos se encuentre bien")
```

```
Buenas tardes Sr. Jhon Lennon, esperemos se encuentre bien
Buenas tardes Sr. Lennon Jhon, esperemos se encuentre bien
```

Formateando cadenas: f-strings

```
nombre, apellido = "Jhon", "Lennon"
mensaje = f"Buenas tardes Sr. {nombre} {apellido}, esperemos se encuentre bien"
print(mensaje)
```

Buenas tardes Sr. Jhon Lennon, esperemos se encuentre bien

- las cadenas f se evalúan en tiempo de ejecución, puede incluir todas y cada una de las expresiones válidas de Python en ellas.

```
nombre = "Luis Humberto"
profesion = "Ingeniero Industrial"
aficion = "jugar Ajedrez"
trabajas = "Universidad Peruana de Ciencias Aplicadas"
mensaje = (f"Hola {nombre}. Eres de profesión {profesion}, "
           f"te gusta {aficion} y trabajas en la {trabajas}")
print(mensaje.upper())
```

HOLA LUIS HUMBERTO. ERES DE PROFESIÓN INGENIERO INDUSTRIAL, TE GUSTA JUGAR AJEDREZ Y TRABAJAS EN LA UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

f-string: Alineamiento

- Usando `f-string`, se puede alinear la salida.

Opción	Significado
<	Obliga a que la expresión dentro de las llaves esté alineada a la izquierda. Este es el predeterminado para las cadenas.
>	Obliga a que la expresión dentro de las llaves esté alineada a la derecha. Este es el predeterminado para los números.
^	Obliga a centrar la expresión dentro de las llaves.

Formateando cadenas: f-strings

- A continuación, se muestra un ejemplo de uso de la alineación tanto para un número como para una cadena. El "|" se utilizará en este ejemplo de f-string para ayudar a delinear el espaciado.
- Asimismo, se presenta el manejo del espacio en el que se imprimirá la salida. El número después de ":" hará que ese campo tenga ese número de caracteres de ancho.

```
variable = 3.141592653589793
print(f"Usando una variable numérica = {variable}")
print(f"|{variable:25}|")   # Los números se alinean por defecto al lado derecho
print(f"|{variable:<25}|")  # Esto alinea a la izquierda
print(f"|{variable:^25}|")  # Esto centra la salida
```

```
Usando una variable numérica = 3.141592653589793
|        3.141592653589793|
|3.141592653589793         |
|    3.141592653589793     |
```

Formateando cadenas: f-strings

- Ejemplo usando variable numérica

```
variable = "Python 3.12.4"
print(f"Usando una variable alfanumérica = {variable}")
print(f"|{variable:25}|")   # Los textos se alinean por defecto a la izquierda
print(f"|{variable:>25}|")  # Esto alinea a la derecha
print(f"|{variable:^25}|")  # Esto centra la salida
```

```
Usando una variable alfanumérica = Python 3.12.4
|Python 3.12.4             |
|            Python 3.12.4|
|      Python 3.12.4       |
```

Formateando cadenas: f-strings

- Ejemplo poniendo comas y puntos decimal. La letra “d” se usa en enteros y “f” en punto flotante

```
variable = 100000
print(f"Esto imprime sin formatear {variable}")
print(f"Esto imprime con formato {variable:,d}")
print(f"Esto imprime con espaciado y formato {variable:10,d}\n")

variable = 1200356.8796
print(f"Con dos decimales: {variable:.2f}")  # f=flotante, .2=dos decimales
print(f"Con cuatro decimales: {variable:.4f}")  # f=flotante, .4=cuatro decimales
print(f"Con dos decimales y coma: {variable:,.2f}")  # ,=formato con miles, millones
```

```
Esto imprime sin formatear 100000
Esto imprime con formato 100,000
Esto imprime con espaciado y formato    100,000

Con dos decimales: 1200356.88
Con cuatro decimales: 1200356.8796
Con dos decimales y coma: 1,200,356.88
```

Formateando cadenas: f-strings

- Otros ejemplos de uso de espaciado, decimales y comas
- Analice y discuta los siguientes ejemplos

```
print(f"Número{' '*3}Cuadrado{' '*4}Cubo")
for x in range(1, 11):
    print(f'{x:3d}{" "*7}{x**2:4d}{" "*6}{x**3:5,d}')
```

```
Número    Cuadrado      Cubo
  1           1           1
  2           4           8
  3           9          27
  4          16          64
  5          25         125
  6          36         216
  7          49         343
  8          64         512
  9          81         729
 10         100       1,000
```

Analice el formateado, por ejemplo:
{x**3:5,d}

- El resultado de x elevado al cubo se pone en un espacio de 5, con comas cuando se pasen los miles
- La letra “d” se usa porque se trata de número enteros

Formateando cadenas: f-strings

- Analice y discuta el siguiente ejemplo

```
print(f"Número{' '*3}Cuadrado{' '*5}Cubo")
for x in range(1, 11):
    x = float(x)
    print(f'{x:5.2f}{" "*4}{x**2:6.2f}{" "*3}{x**3:8,.2f}')
```

```
Número    Cuadrado      Cubo
 1.00      1.00        1.00
 2.00      4.00        8.00
 3.00      9.00       27.00
 4.00     16.00       64.00
 5.00     25.00      125.00
 6.00     36.00      216.00
 7.00     49.00      343.00
 8.00     64.00      512.00
 9.00     81.00      729.00
10.00    100.00    1,000.00
```

Analice el formateado, por ejemplo:
{x**3:8.2f}

- El resultado de x elevado al cubo se pone en un espacio de 8, con comas cuando se pasen los miles
- La letra “f” se usa porque se trata de número de punto flotante
- El .2 es para que ponga dos decimales

PREGRADO

Ingeniería de Sistemas de Información

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC

Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco

Lima 33 - Perú

T 511 313 3333

https://www.upc.edu.pe

exígete, innova

LAUREATE
INTERNATIONAL
UNIVERSITIES