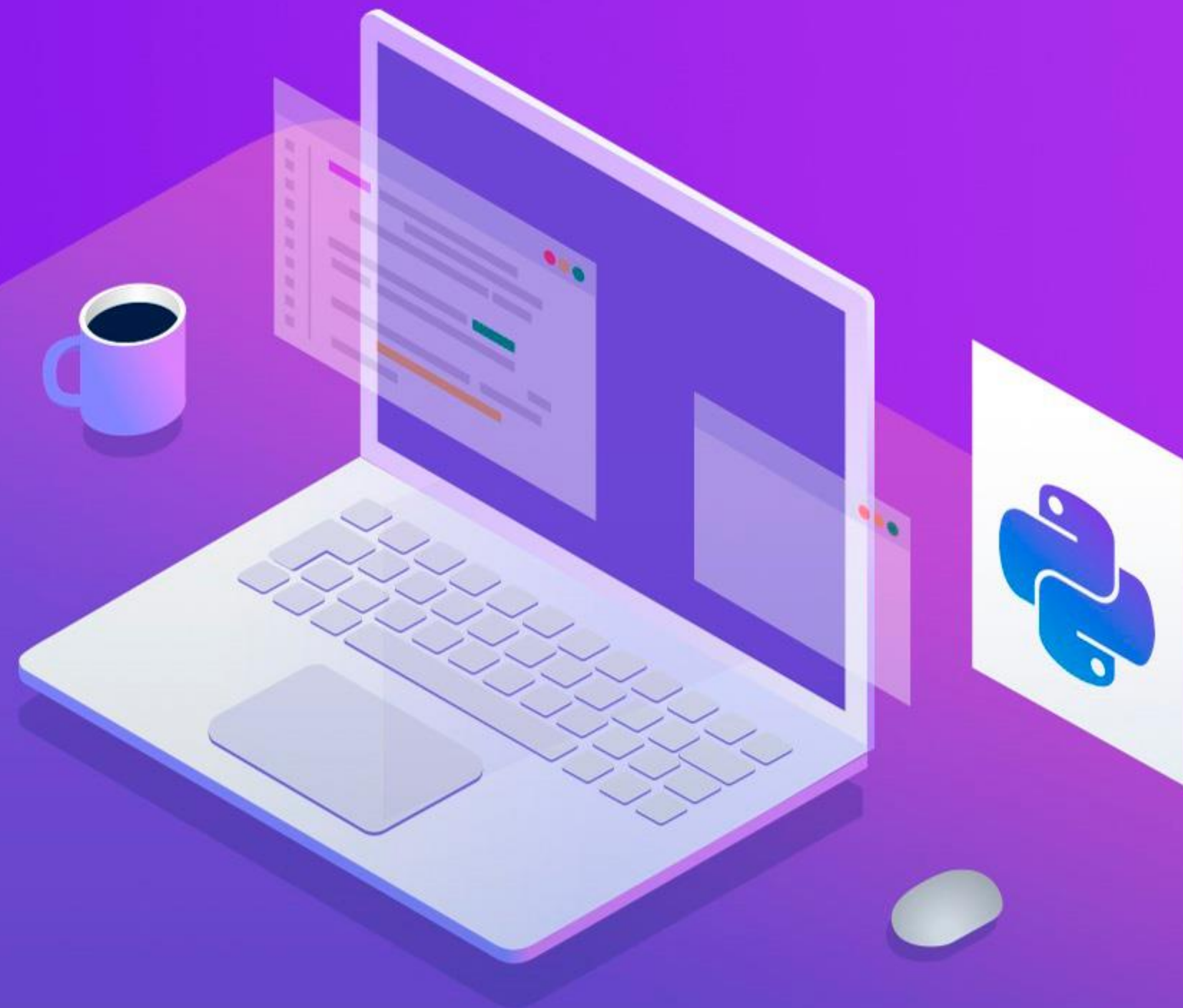




PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

INTRODUCCIÓN A LOS TIPOS DE DATOS Y FUNCIONES

>>> Parte 1: Introducción a los tipos de datos



Recordemos

Ya estudiamos el trabajo más básico con datos, mediante el uso y la operación de variables. Ahora veremos operaciones y herramientas mucho más avanzadas que nos servirán para poder procesar datos de forma mucho más eficiente.

Lo importante es recordar que las variables almacenan datos, y que hay distintos tipos de datos.



Recordemos los tipos de datos

`int`

Variables que contienen números enteros.

CÓDIGO	RESULTADO
<pre>variable_entera=-9 print(variable_entera)</pre>	<pre>-9</pre>

Recordemos los tipos de datos

float

Variables que contienen números decimales.

CÓDIGO	RESULTADO
<pre>variable_decimal=8.75 print(variable_decimal)</pre>	8.75

Recordemos los tipos de datos

string

Variables que contienen información de tipo texto.

CÓDIGO	RESULTADO
<pre>variable_texto="este es un texto de ejemplo" print(variable_texto)</pre>	<pre>este es un texto de ejemplo</pre>

Recordemos los tipos de datos

bool

Las variables `bool` pueden almacenar solo dos datos **True** y **False**.

Uno puede declararlas explícitamente.

CÓDIGO

```
variable_tipo_bool = True  
variable_tipo_bool2 = False
```

O bien pueden ser el resultado de una operación lógica.

CÓDIGO

```
numero = 2  
  
variable_bool1 = numero != 2  
variable_bool2 = numero == 2  
variable_bool3 = variable_bool1 and  
variable_bool2
```

Nuevas variables: listas

Variables tipo listas

- Pueden almacenar varios datos a la vez.
- No tenemos operadores para poder trabajar con variables de tipo texto y listas.

```
17     vecs = [None] * npoints
18
19     for i in range(npoints):
20         start = i * self.ndims
21         end = start + self.ndims
22         vecs[i] = pdata[start:end]
23
24     return [x[1] for x in index_name_array], vecs
25
26 def shape(self):
27     return [len(self.names), self.ndims]
28
29 def toPyModel(file_ptr):
30     if bool(file_ptr) == False:
31         raise ValueError("Null pointer")
32     m = file_ptr.contents
33     m.__createfrom__ = 'C'
34     return m
35
36 def load_file_to_memory(datadir, ndims):
37     file_name = os.path.join(datadir, "feature.bin")
38     idfile = os.path.join(datadir, "id.txt")
39     names = [x.strip() for x in str.split(open(idfile).read())]
40     memfile = libbigfile.load_file(file_name, ndims, len(names))
41     if not memfile:
42         print("[load_file_to_memory] failed to open %s" % file_name)
43         return None
44     file = toPyModel(memfile)
45     file.load_ids(ndims, names)
46     return memfile
```


Funciones

- Por eso es importante definir el concepto de **funciones**. Python permite la operación con strings y listas mediante funciones.
- Además, existen varias funciones que permiten ampliar nuestro espectro de operaciones con variable enteras y decimales. Muchas de estas funciones ya están integradas en Python.

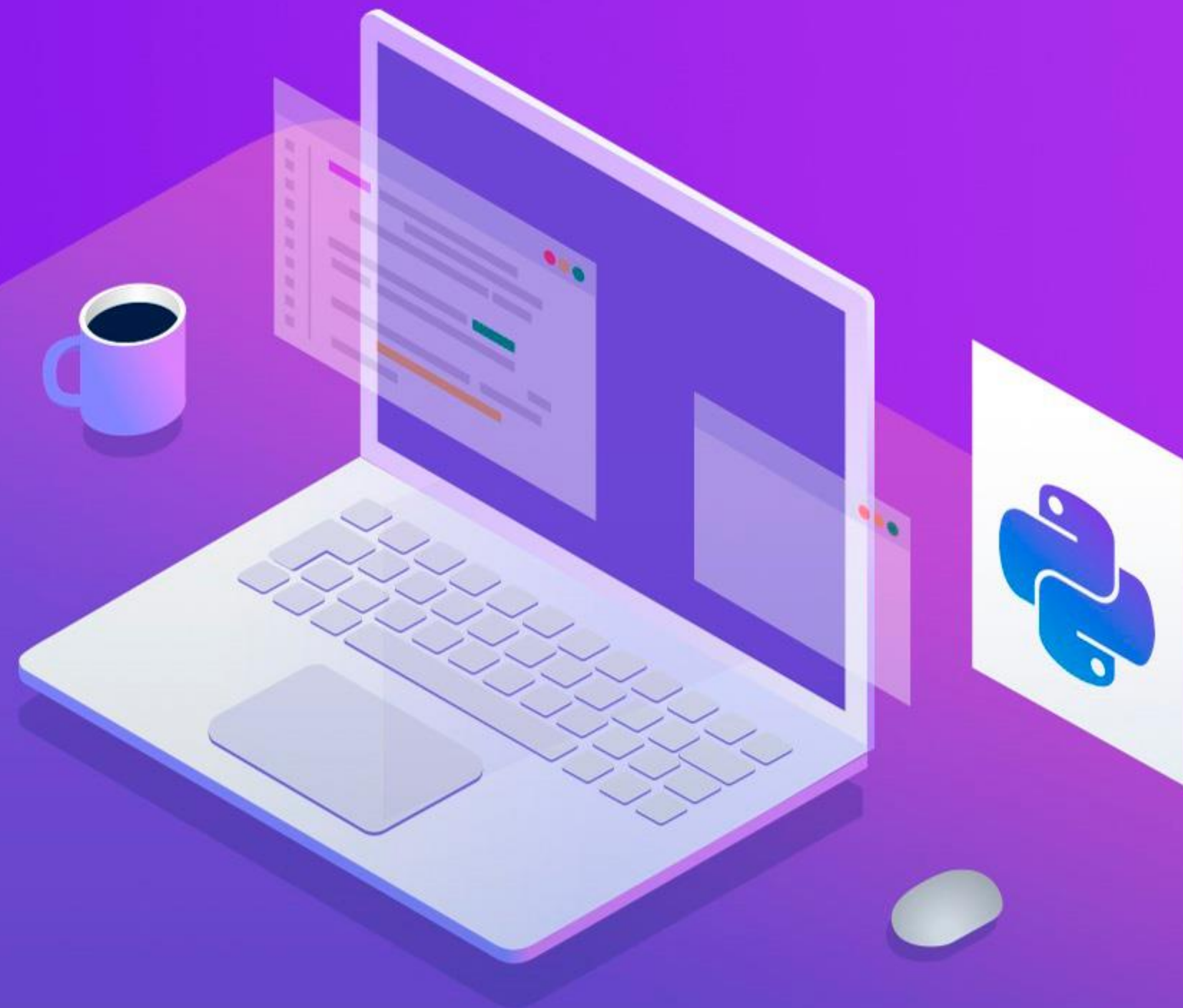
```
17     vecs = [None] * npoints
18
19     for i in range(npoints):
20         start = i * self.ndims
21         end = start + self.ndims
22         vecs[i] = pdata[start:end]
23
24     return [x[1] for x in index_name_array], vecs
25
26 def shape(self):
27     return [len(self.names), self.ndims]
28
29 def toPyModel(file_ptr):
30     if bool(file_ptr) == False:
31         raise ValueError("Null pointer")
32     m = file_ptr.contents
33     m.__createfrom__ = 'C'
34     return m
35
36 def load_file_to_memory(datadir, ndims):
37     file_name = os.path.join(datadir, "feature.bin")
38     idfile = os.path.join(datadir, "id.txt")
39     names = [x.strip() for x in str.split(open(idfile).read())]
40     memfile = libbigfile.load_file(file_name, ndims, len(names))
41     if not memfile:
42         print("[load_file_to_memory] failed to open %s" % file_name)
43         return None
44     file = toPyModel(memfile)
45     file.load_ids(ndims, names)
46     return memfile
```




PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

INTRODUCCIÓN A LOS TIPOS DE DATOS Y FUNCIONES

>>> Parte 2: Funciones



Funciones

Recordemos este ejercicio:

CÓDIGO

```
import random
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue:
"+str(numero_dado)+"\n")

if numero_dado%2==0:
    print("Este es un número par. \n")
else:
    print("Este es un número impar. \n")
```

En las dos primeras líneas de código, lo que hicimos fue generar un número aleatorio entre 1 y 6. Para eso, en la línea 1 "importamos" el paquete random.

Después de importar el paquete, podemos generar el número aleatorio.

Funciones

Analicemos esta línea de código:

CÓDIGO

```
import random
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue:
"+str(numero_dado)+"\n")

if numero_dado%2==0:
    print("Este es un número par. \n")
else:
    print("Este es un número impar. \n")
```

1

```
numero_dado=random.randint(1,6)
```

2

random.randint(1,6) genera un número aleatorio entre 1 y 6, y este número lo asignamos a la variable numero_dado.

Funciones

Analicemos esta línea de código:

CÓDIGO

```
import random
numero_dado=random.randint(1,6)
print("El número que arrojó el dado fue:
"+str(numero_dado)+"\n")

if numero_dado%2==0:
    print("Este es un número par. \n")
else:
    print("Este es un número impar. \n")
```

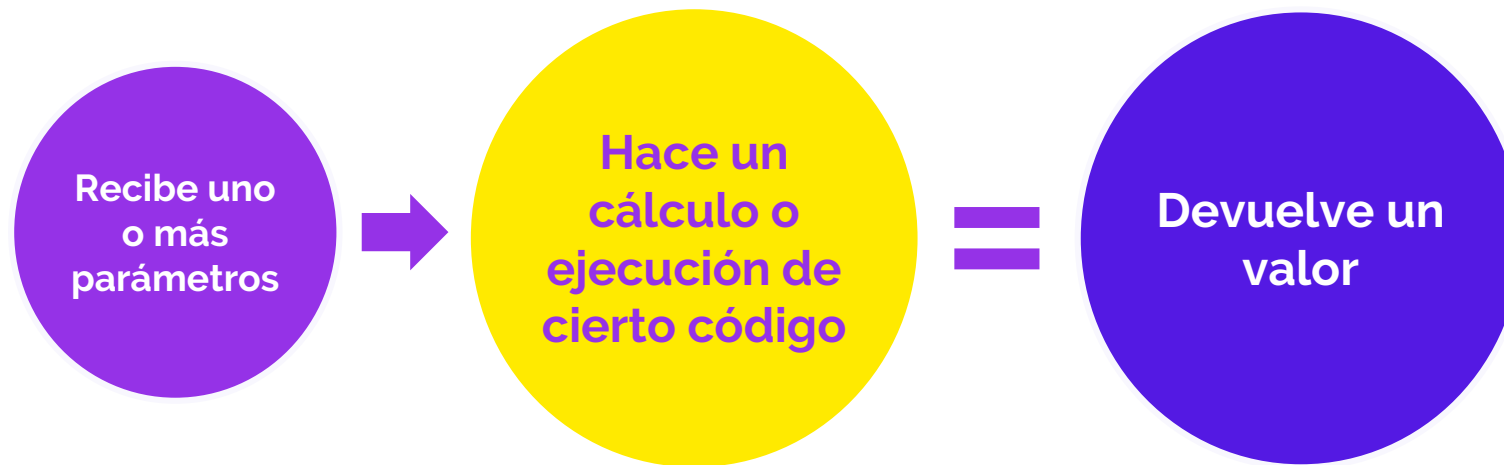
3

No obstante, sabemos que `random` representa al recientemente importado paquete "random". Por lo tanto ¿Qué es `".randint(1,6)"`?

4

Esto no es simplemente una secuencia de caracteres o un comando, sino que es una **función**. El "." entre `random` y `randint` nos indica que este último es una función del paquete `random`.

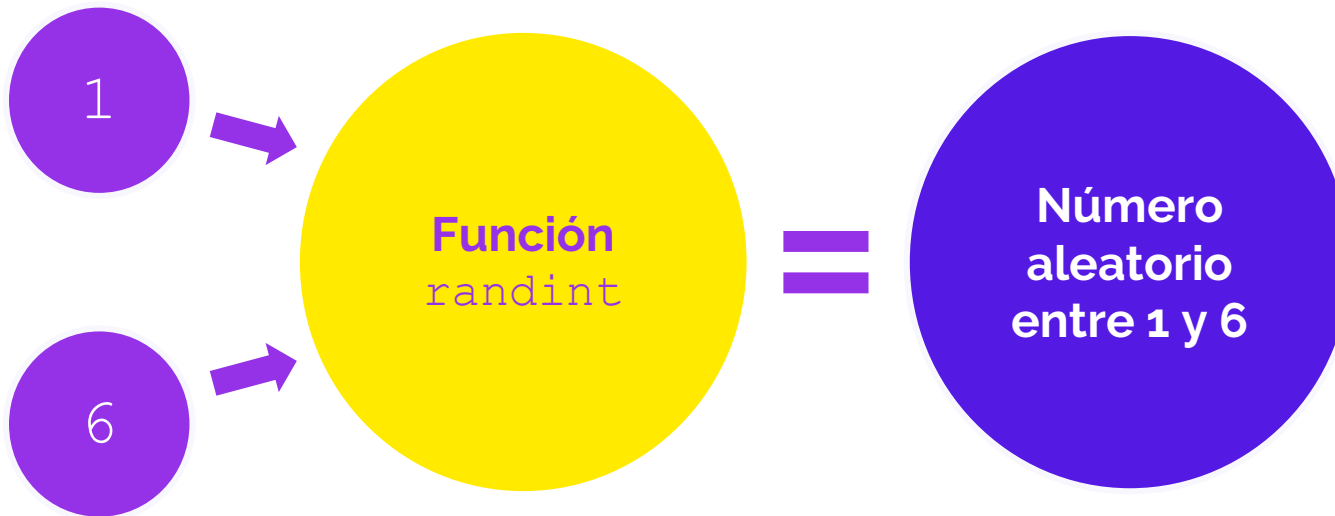
¿Cómo podríamos caracterizar a una función?



¿Cómo podríamos caracterizar a una función?

`randint(1, 6)`

Los parámetros de la función son 1 y 6. La función se llama "randint".
El valor retornado es un número entre 1 y 6.



¿Cómo podemos definir las funciones en Python?

```
def nombre_funcion(a,b,c):  
    #ejecución de cierto código  
    return (valor retornado)
```

La definición puede ser un poco compleja de entender de forma genérica, por lo tanto, veamos un ejemplo.

¿Cómo podemos definir las funciones en Python?

Recuerda revisar la Ruta de ejercicios.
Ejercicio EM2-07 →

Digamos que queremos crear una función que sume dos números. Para poder hacerlo, ocuparemos el siguiente código:

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Analicemos cada una de las partes de este ejemplo

Comando `def`

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

El comando `def` sirve para que Python sepa cuando estamos definiendo una función. **Siempre** se debe ocupar este comando **al definir una función**.

Analicemos cada una de las partes de este ejemplo

Agregar nombre de la variable

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Después del comando `def` y separada por un espacio **se agrega el nombre de la variable**. El nombre queda completamente a elección del usuario, **pero no se pueden ocupar palabras que ya sean comandos de Python**. Por ejemplo, no podría llamar a una función `def`, ni tampoco `if`, `else`, `while`, `int`, `print`, `input`, etc.

Analicemos cada una de las partes de este ejemplo

Definir los parámetros

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Éstos son sumamente importantes, ya que son el valor de entrada de la función, y sobre ellos se operará posteriormente. Algunas consideraciones:

- No hay un límite para la cantidad de parámetros que se quiera incluir en la función. Asimismo, también se puede crear una función sin parámetros.

Analicemos cada una de las partes de este ejemplo

Definir los parámetros

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Éstos son sumamente importantes, ya que son el valor de entrada de la función, y sobre ellos se operará posteriormente. Algunas consideraciones:

- Los parámetros pueden tener cualquier nombre.

Analicemos cada una de las partes de este ejemplo

Definir los parámetros

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Éstos son sumamente importantes, ya que son el valor de entrada de la función, y sobre ellos se operará posteriormente. Algunas consideraciones:

- Para definir, cada parámetro, se deben incluir entre paréntesis y separar por comas. Si se quiere crear una función sin parámetros, entonces se escribe el paréntesis sin nada adentro.

Analicemos cada una de las partes de este ejemplo

Respetar la indentación

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Al igual que en `if` o `while`, es muy importante respetar la **indentación**. Si no se respeta, entonces Python no sabrá que el código que se está escribiendo está "dentro" de esta función.

Esto es particularmente sensible al ocupar los parámetros, ya que estos **solo** existen dentro de la función, y no se pueden ocupar fuera.

Analicemos cada una de las partes de este ejemplo

Respetar la indentación

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Por eso, es una buena práctica considerar a las funciones como "cajas", donde al escribir código dentro de la función solo se pueden ocupar las cosas que están dentro de esta caja. Asimismo, fuera de esta caja no podemos ocupar elementos que estén dentro de ella.

Analicemos cada una de las partes de este ejemplo

Comando `return`

CÓDIGO

```
def suma_de_dos_numeros(num1, num2):  
    variable_con_valor_retornado = num1 + num2  
    return variable_con_valor_retornado
```

Finalmente, definimos el valor retornado mediante el comando `return`. Este comando nos dice que la función “devolverá” lo que esté después del comando `return`. En general, se devuelve una variable que tendrá algún tipo de dato dentro.

Analicemos cada una de las partes de este ejemplo

Recuerda revisar la Ruta de ejercicios.
Ejercicio EM2-08 →

Comando `return`

CÓDIGO

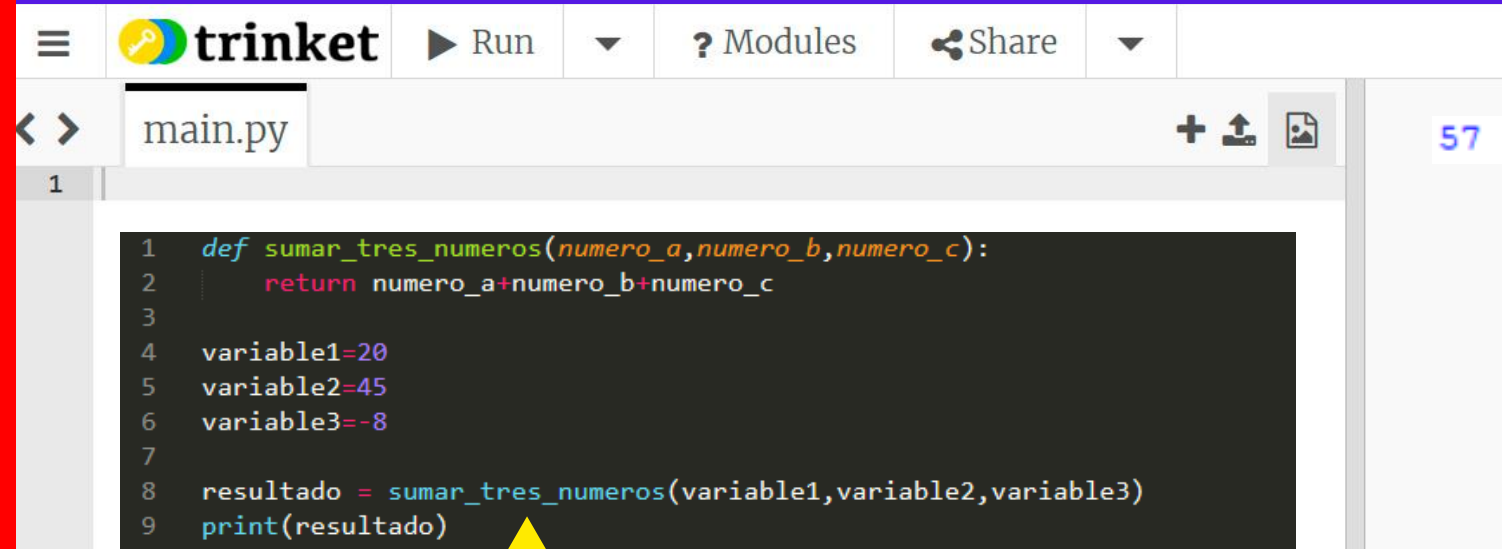
```
def suma_de_dos_numeros(num1, num2):  
    return num1 + num2
```

También se puede operar directamente en el `return`, como pueden ver en este ejemplo.

¿Queremos crear una función que reciba tres números y los sume?

Recuerda revisar la Ruta de ejercicios.

Ejercicio EM2-09 →

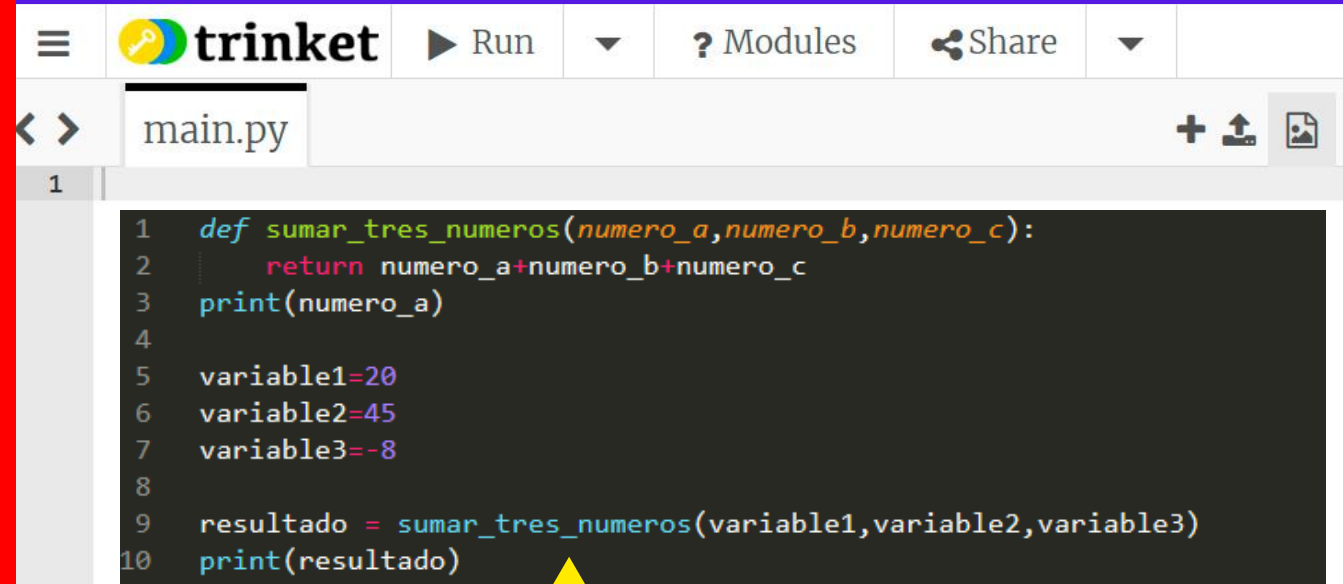


```
1 def sumar_tres_numeros(numero_a,numero_b,numero_c):
2     return numero_a+numero_b+numero_c
3
4 variable1=20
5 variable2=45
6 variable3=-8
7
8 resultado = sumar_tres_numeros(variable1,variable2,variable3)
9 print(resultado)
```

En este ejemplo, **definimos la función previamente**, y luego la usamos llamándola por su nombre y entregándole parámetros. En este caso, el valor retornado lo almacenamos en la variable resultado que posteriormente imprimimos en consola.

¿Queremos crear una función que reciba tres números y los sume?

Recuerda revisar la Ruta de ejercicios.
Ejercicio EM2-10 →



```
1 def sumar_tres_numeros(numero_a,numero_b,numero_c):
2     return numero_a+numero_b+numero_c
3 print(numero_a)
4
5 variable1=20
6 variable2=45
7 variable3=-8
8
9 resultado = sumar_tres_numeros(variable1,variable2,variable3)
10 print(resultado)
```

NameError: name 'numero_a' is not defined

En este ejemplo, al intentar ejecutar este código Python arroja un error. Esto es porque en la línea 3 estamos intentando imprimir la variable `numero_a` pero esta solo existe dentro de la función `sumar_tres_numeros`

Veamos una modificación al ejemplo anterior

Si una función que multiplica tres números, y otra que suma dos números y los divide por dos (es decir, saca el promedio)

Recuerda revisar la Ruta de ejercicios.
Ejercicio EM2-11 →

```
def multiplicar_tres_numeros(numero_a,numero_b,numero_c):  
    return numero_a*numero_b*numero_c  
  
def sacar_promedio_entre_dos(primer_numero,segundo_numero):  
    return (primer_numero+segundo_numero)/2  
  
variable1=2  
variable2=4  
variable3=1  
  
resultado1 = multiplicar_tres_numeros(variable1,variable2,variable3)+sacar_promedio_entre_dos(variable1,variable2)  
print(resultado1)  
  
resultado2 = sacar_promedio_entre_dos(multiplicar_tres_numeros(variable1,variable2,variable3),variable2)  
print(resultado2)
```

11.0
6.0

Analicemos los resultados

Para resultado1

CÓDIGO

```
resultado1 = multiplicar_tres_numeros(variable1,variable2,variable3)+sacar_promedio_entre_dos(variable1,variable2)
print(resultado1)
```

Se suma el valor retornado de la función

`multiplicar_tres_numeros(variable1,variable2,variable3)`, que es 8.

Con el resultado de la función `sacar_promedio_entre_dos(variable1,variable2)`, que es 3.

La suma de ambos, como se puede ver por el resultado en consola, es 11.

Analicemos los resultados

Para resultado2

CÓDIGO

```
resultado2 = sacar_promedio_entre_dos(multiplicar_tres_numeros(variable1,variable2,variable3),variable2)
print(resultado2)
```

Recordemos que el valor retornado de la función `multiplicar_tres_numeros(variable1,variable2,variable3)`, es 8.

Este valor entra como el primer parámetro a la función `sacar_promedio_entre_dos(multiplicar_tres_numeros(variable1,variable2,variable3),variable2)`

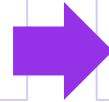
Por lo tanto, dentro de la función `sacar_promedio_entre_dos(multiplicar_tres_numeros(variable1,variable2,variable3),variable2)` se saca el promedio entre 8 y 4, que es 6. Este es el resultado que podemos ver en la consola.

Consideraciones importantes sobre las funciones

Las funciones se ocupan en general para hacer nuestro código más eficiente.



Si es que estamos repitiendo el mismo pedazo de código, muchas veces probablemente sea mejor hacer una función.



Varios de los conceptos que se ven al programar en Python incluyen funciones dentro de funcionamiento.



En particular, para poder aprender a programar usando `strings` y listas, tendremos que ocupar algunas de sus funciones.

Cierre

Has finalizado la revisión de los contenidos que corresponden a esta clase.

A continuación, te invitamos a estudiar la siguiente clase del módulo.