

Pensamiento Computacional

Aldana Rastrelli, Juan Pablo Bulacios, Pablo Notari

2023-12-27

Table of contents

Pensamiento Computacional	4
Docentes de la Cátedra	4
La Materia	5
Fundamentación	5
Objetivos Generales	5
1 Introducción a la Algoritmia y a la Programación	7
1.1 Introducción	7
1.1.1 La Computadora	7
1.1.2 Software y Hardware	7
1.1.3 Sistema Operativo	8
1.1.4 Algoritmo	8
1.1.5 Programa	8
1.1.6 Lenguaje de Programación	9
1.1.7 Entorno de Desarrollo	9
1.2 Lenguaje Python	9
1.2.1 Hola, Mundo!	10
1.3 Anexo: Replit	10
1.3.1 Creación de una nueva cuenta	10
1.3.2 Creación de un nuevo proyecto	12
1.3.3 Uso del nuevo proyecto	13
2 Tipos de Datos, Expresiones y Funciones	18
2.1 Sentencias Básicas	18
2.1.1 Flujo de Control de un Programa	18
2.1.2 Valores y Tipos	19
2.1.3 Variables	21
2.1.4 Funciones	22
2.1.5 Ingreso de Datos por Consola	26
2.2 Buenas Prácticas de programación	28
2.2.1 Sobre Comentarios	28
2.2.2 Sobre Convención de Nombres	28
2.2.3 Sobre Ordenamiento de Código	29
2.2.4 Sobre uso de Parámetros en Funciones	29

2.3	Tipos de Datos	31
2.3.1	Datos Simples	31
2.3.2	Operadores Numéricos	32
2.3.3	Operadores de Texto	33
2.3.4	Input y Casteo	34
2.4	Bonus Track: Algunas Funciones Predefinidas de Python	37
3	Estructuras de Control	39
3.1	Subtitle	39
4	Tipos de Datos Estructurados	40
4.1	Subtitle	40
5	Entrada y Salida de Información	41
5.1	Subtitle	41
6	Bibliotecas	42
6.1	Subtitle	42
	Referencias	43
	Guía de Ejercicios	44
	Guía 1: Introducción a la Algoritmia y la Programación	44
	Guía 2: Tipos de Datos, Expresiones y Funciones	45
	Variables y Tipos de Dato	45

Pensamiento Computacional

Bienvenidos y bienvenidas a la cátedra de Pensamiento Computacional del Ciclo Básico Común de la Facultad de Ingeniería - UBA.

Docentes de la Cátedra

- **Prof. Titular:** Méndez, Mariano
- Bulacios, Juan Pablo
- Notari, Pablo
- Rastrelli, Aldana

La Materia

Fundamentación

El pensamiento computacional es una disciplina que ha sido definida como “el conjunto de procesos de pensamiento implicados en la formulación de problemas y sus soluciones, de manera que dichas soluciones sean representadas de una forma que puedan ser efectivamente ejecutadas por un agente de procesamiento de información”, entendiendo por esto último a un humano, una máquina o una combinación de ambos.

Reconoce antecedentes en trabajos de la Carnegie Mellon University de la década de 1960 y del Massachusetts Institute of Technology de alrededor de 1980, aunque su auge en la educación superior llegó con la primera década del siglo XXI.

Las herramientas básicas en las que se funda el pensamiento computacional son la descomposición, la abstracción, el reconocimiento de patrones y la algoritmia. Está ampliamente aceptado que estas herramientas no sirven solamente a los profesionales de Ciencias de la Computación y de Informática, sino a cualquier persona que deba resolver problemas, con lo cual el pensamiento computacional deviene una técnica de resolución de problemas. Actualmente, los y las profesionales de la Ingeniería requieren de una capacidad analítica que les permita resolver problemas, y en ese sentido el pensamiento computacional se convierte en un soporte invaluable de esa competencia (cada vez más las ciencias de la computación y la informática constituyen una ciencia básica para todas las ingenierías).

Si bien el pensamiento computacional no necesariamente requiere del uso de computadoras, la programación de computadoras se convierte en su complemento ideal. En primer lugar, porque permite comprobar, mediante la codificación de un algoritmo en un programa, la validez de la solución encontrada al problema, de manera sencilla y prácticamente inmediata. En segundo lugar, porque la programación incentiva la creatividad, la capacidad para la autoorganización y el trabajo en equipo. En tercer lugar, porque la programación constituye un recurso habitual del trabajo en el campo profesional de la ingeniería.

Objetivos Generales

El objetivo general de la asignatura es que los/as estudiantes adquieran habilidades de resolución de problemas de ingeniería mediante el soporte de un lenguaje de programación multi-

paradigma.

1 Introducción a la Algoritmia y a la Programación

1.1 Introducción

Como en todas las disciplinas, la Ingeniería de Software y la Programación de Sistemas en general tienen un **lenguaje técnico** específico. La utilización de ciertos términos y el compartir de ciertos conceptos agiliza el diálogo y mejora la comprensión con los pares.

En este capítulo vamos a hacer una breve introducción de ciertos conceptos, ideas y modelos que van a permitirnos establecer acuerdos y manejar un lenguaje común.

1.1.1 La Computadora

Una computadora es un dispositivo físico de procesamiento de datos, con un propósito general. Todos los programas que escribiremos serán ejecutados (o *corridos*) en una computadora. Una computadora es capaz de procesar datos y obtener nueva información o resultados.

1.1.2 Software y Hardware

Toda computadora funciona con software y hardware. El software es el conjunto de herramientas abstractas (programas), y se le llama **componente lógica** del modelo computacional. El hardware es el **componente físico** del dispositivo. Básicamente, el software dice qué hacer, y el hardware lo hace.

💡 ¿Es indispensable tener una computadora para crear un algoritmo?

La respuesta, sorprendentemente, es no: muchos de los algoritmos que se utilizan de forma computacional hoy en día fueron diseñados varias décadas atrás. Pero la implementación de un algoritmo depende del grado de avance del hardware y la tecnología disponible.

1.1.3 Sistema Operativo

El sistema operativo es el programa encargado de administrar los recursos del sistema. Los recursos (como la memoria, por ejemplo) son disputados entre diferentes programas o procesos ejecutándose al mismo tiempo. El sistema operativo es el que decide cómo administrar y asignar los recursos disponibles.

Los sistemas operativos más comunes el día de hoy son: Windows, Linux, iOS, Android; por ejemplo.

1.1.4 Algoritmo

Un algoritmo es una serie finita de pasos precisos para alcanzar un objetivo.

- “serie”: porque son continuados uno detrás del otro, de forma ordenada.
- “finita”: porque no pueden ser pasos infinitos, en algún momento deben terminar.
- “pasos precisos”: porque en un algoritmo se debe ser lo más específico posible.

Ejemplo Un algoritmo puede ser una receta de cocina: tiene una serie finita de pasos (son ordenados, uno detrás de otro, finitos porque en algún momento deben terminar), que son precisos (porque tienen indicaciones de cuánto agregar de cada ingrediente, cómo incorporarlo a la preparación, etc) y están orientados en alcanzar un objetivo (obtener una comida en particular).

1.1.4.1 Creación de un Algoritmo

La forma en la que trabajaremos la creación de un algoritmo es siguiendo los siguientes pasos: 1. Análisis del problema: entender el objetivo y los posibles casos puntuales del mismo. 2. Primer borrador de solución: confeccionar una idea generalizada de cómo podría resolverse el problema. 3. División del problema en partes: dividir el problema en partes ayuda a descomponer un problema complejo en varios más sencillos. 4. Ensamble de las partes para la versión final del algoritmo: acoplar todo el conjunto de partes del problema para lograr el objetivo general.

Estos cuatro pasos podrán iterarse (repetirse) la cantidad de veces que sean necesarios, para poder lograr acercarnos más a la solución en cada iteración.

1.1.5 Programa

Un programa es un algoritmo escrito en un lenguaje de programación.

1.1.6 Lenguaje de Programación

Un lenguaje de programación es un **protocolo de comunicación**.

Un protocolo es un **conjunto de normas consensuadas**.

⇒ Entonces, un lenguaje de programación es un conjunto de normas consensuadas, entre la persona y la máquina, para poder comunicarse.

Cuando logramos que un *lenguaje* pueda ser comprendido por el humano y por la máquina, tenemos una comunicación efectiva en donde podremos hacer programas y pedirle a la máquina que los ejecute.

Un buen ejemplo de cómo una computadora interpreta nuestras instrucciones sin pensar al respecto, sin tener sentido común y sin ambigüedades, es [este video](#). La computadora lo único que hace es *interpretar* de forma explícita lo que nosotros le pedimos que haga.

Un lenguaje de programación tiene reglas estrictas que se deben respetar y no se admiten ambigüedades o sobreentendidos.

1.1.7 Entorno de Desarrollo

Un entorno de desarrollo es un conjunto de herramientas que nos permiten escribir, editar, compilar y ejecutar programas.

En la materia utilizaremos un entorno de desarrollo llamado Replit, que nos permite escribir código en un editor de texto, compilarlo y ejecutarlo en un mismo lugar de forma online. Pero existen muchos otros entornos de desarrollo, como por ejemplo Visual Studio Code, Eclipse, NetBeans, etc.

1.2 Lenguaje Python

En este curso utilizaremos el lenguaje de programación **Python**. Python es un lenguaje de programación de propósito general, que se utiliza en muchos ámbitos de la industria y la academia.

Python es un lenguaje realmente fácil de aprender, con una curva de aprendizaje muy suave. Es un lenguaje de alto nivel, lo que significa que es un lenguaje que se asemeja mucho al lenguaje natural, y que no requiere de conocimientos de bajo nivel para poder utilizarlo.

1.2.1 Hola, Mundo!

El primer programa que se escribe en cualquier lenguaje de programación es el programa “Hola, Mundo!”. Este programa es un programa que imprime en pantalla el texto “Hola, Mundo!”.

En Python, el programa “Hola, Mundo!” se escribe de la siguiente forma:

```
print("Hola, Mundo!")
```

Hola, Mundo!

`print` es una función que imprime en pantalla el texto que se le pasa entre paréntesis. En este caso, el texto que se le pasa como parámetro es `"Hola, Mundo!"`. Al escribir las comillas dobles, estamos indicando que el texto que se encuentra entre ellas es un texto literal.

De la misma forma, podremos imprimir cualquier otro mensaje en pantalla, como por ejemplo:

```
print("Hola, me llamo Rosita y soy programadora")
```

Hola, me llamo Rosita y soy programadora

Al igual que Rosita, al hacer nuestro primer ‘Hola, Mundo!’ nos convertimos en programadores. ¡Felicitaciones!

A partir de la próxima clase, comenzaremos a ver cómo escribir programas más complejos, que nos permitan resolver problemas más interesantes.

1.3 Anexo: Replit

1.3.1 Creación de una nueva cuenta

Para utilizar replit vamos a ingresar a <https://replit.com/>.

Vamos a presionar luego en **Sign Up**, donde va a pedir crear una cuenta, o iniciar sesión si ya tenemos una. Una de nuestras opciones es, si tenemos una cuenta google ya creada, iniciar sesión con eso. De lo contrario, podemos crear una cuenta nueva con un mail.

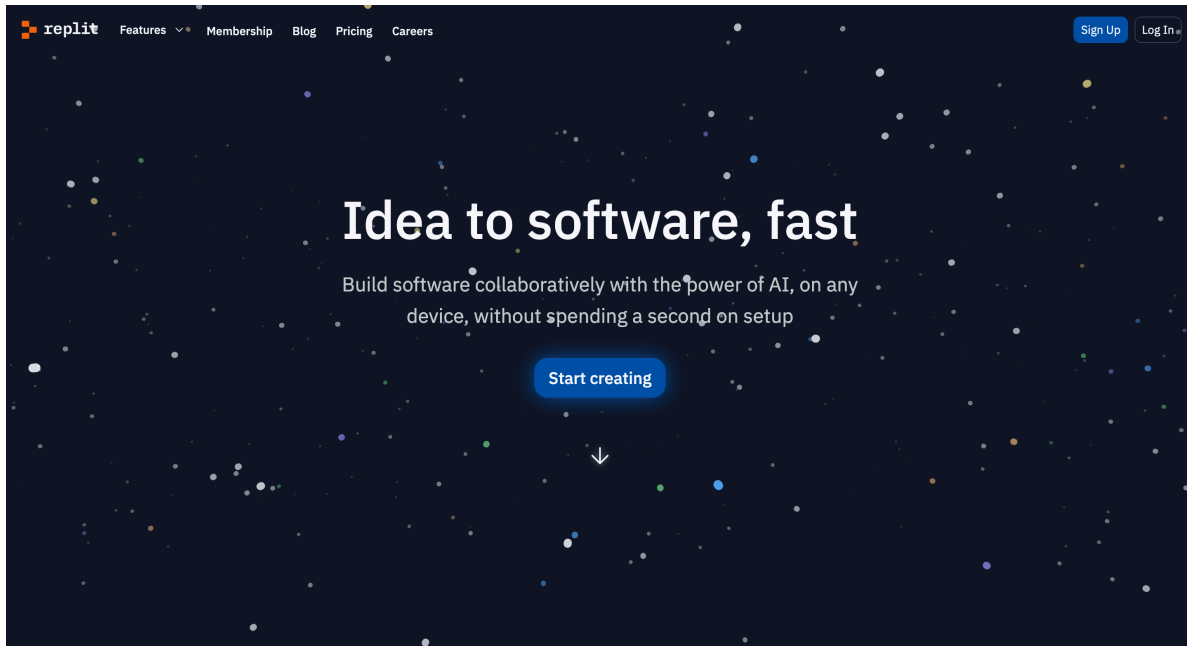


Figure 1.1: Página de inicio de Replit

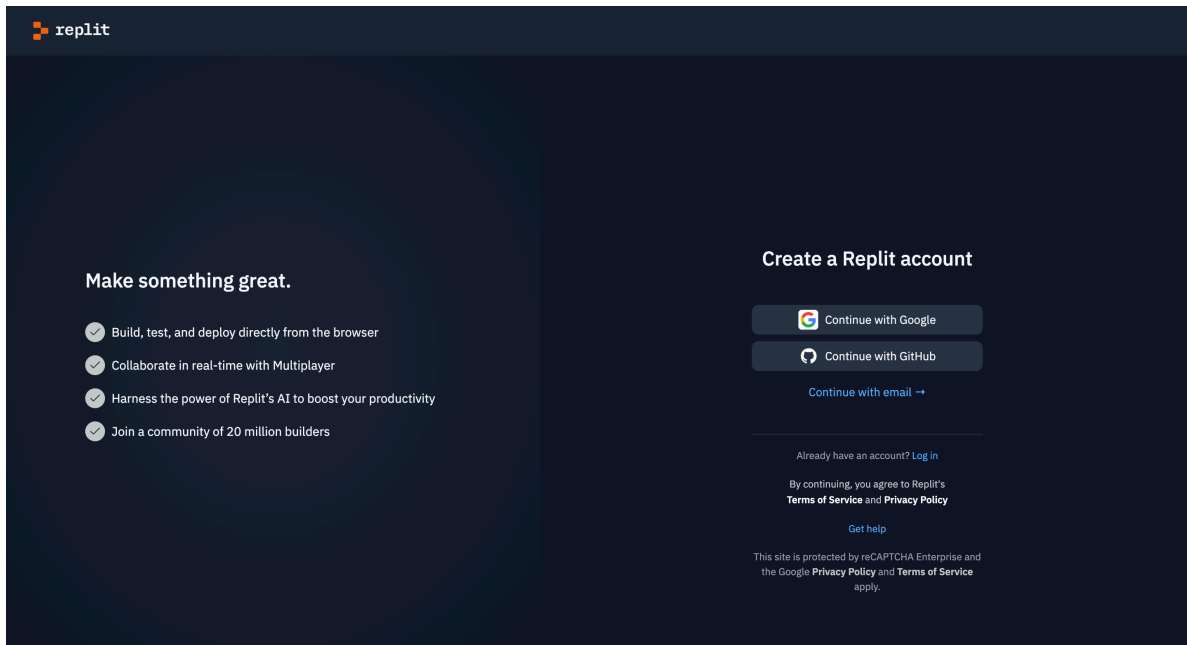


Figure 1.2: Página de creación de cuenta de Replit

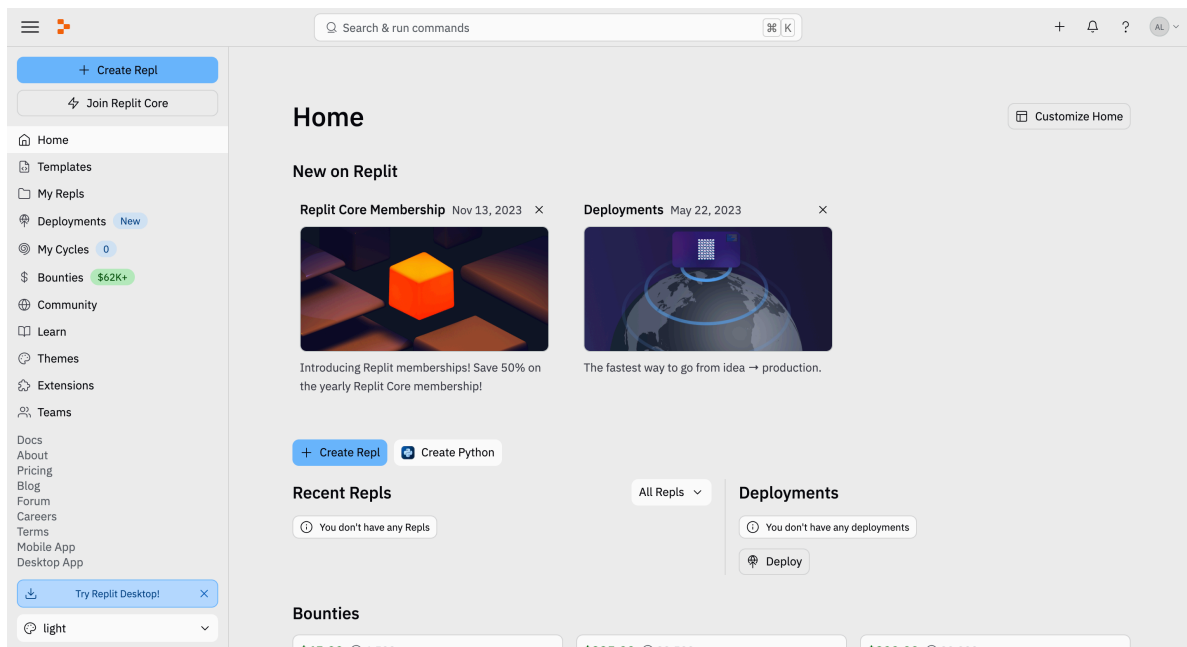


Figure 1.3: Home de Replit

1.3.2 Creación de un nuevo proyecto

Una vez creada la cuenta e iniciado sesión, vamos a ver esta pantalla:

En la misma vamos a ver muchas opciones, pero la que nosotros nos interesa es el botón de **+ Create Repl**, que nos va a permitir crear un nuevo proyecto.

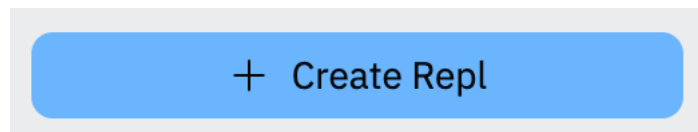
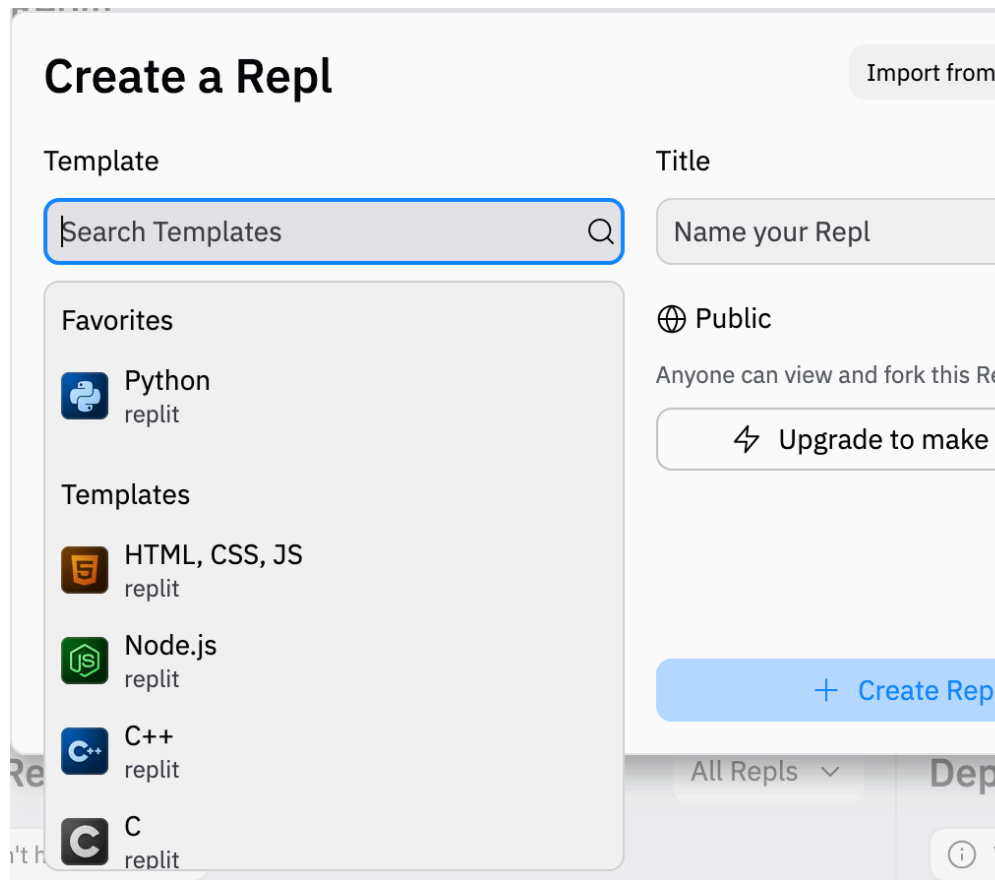


Figure 1.4: Botón de creación de un nuevo proyecto en Replit



Se va a abrir la siguiente ventana:

Donde vamos a buscar y elegir en “Templates” el lenguaje de programación Python. Luego, vamos a asignarle un nombre y seleccionar “Create repl”.

Se debería ver algo así:

1.3.3 Uso del nuevo proyecto

Los espacios o proyectos en replit se llaman **Workspace**, que significa **espacio de trabajo**. En este espacio de trabajo vamos a poder escribir código, ejecutarlo, y ver los resultados de la ejecución.

Una vez creado el espacio de trabajo, se nos va a abrir una pantalla donde vamos a ver varias cosas.

Inicialmente, tenemos en el centro el espacio de edición de código, donde vamos a escribir nue-

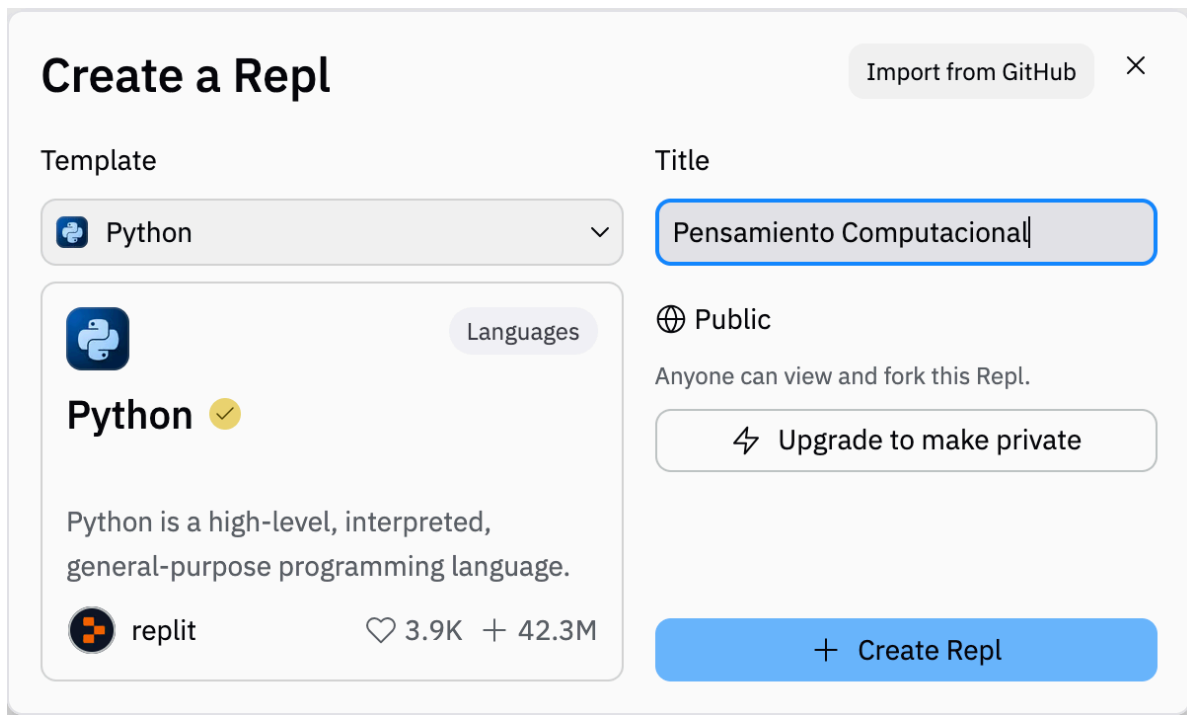
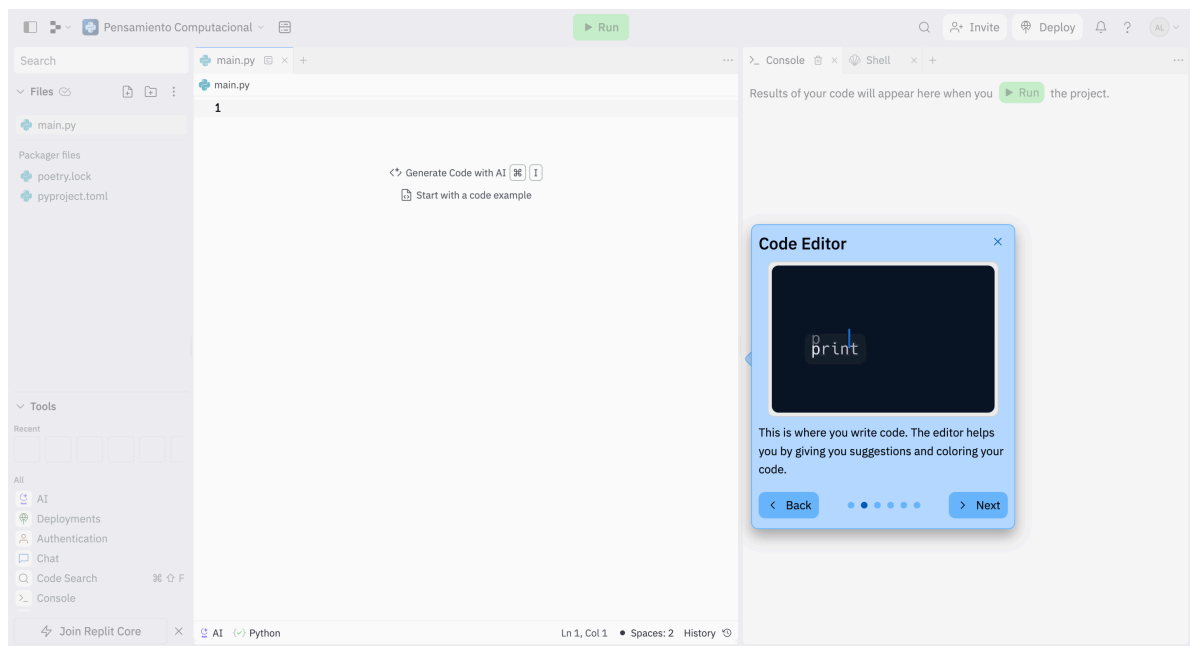


Figure 1.5: Ventana completa de creación de un nuevo proyecto en Replit



stro programa.

En la parte superior, vamos a ver un botón de Run, que nos va a permitir ejecutar el programa

que escribimos.

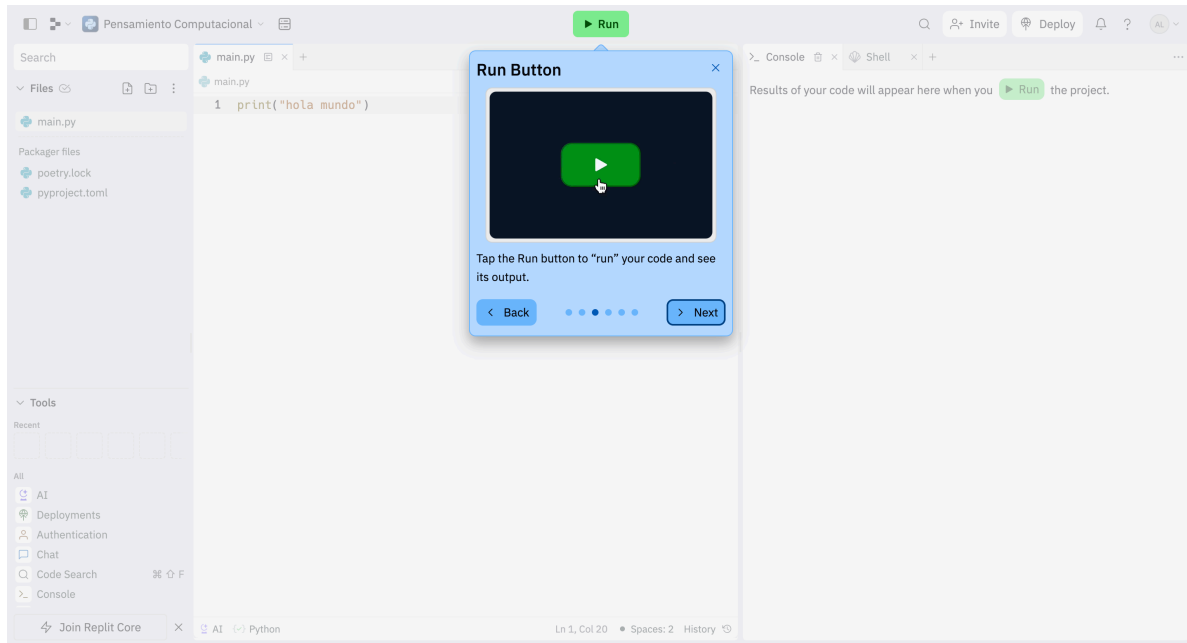


Figure 1.6: Botón de ejecución de código

En la parte derecha, vamos a ver el resultado de la ejecución del programa. En este caso, como no escribimos nada, no hay nada para mostrar.

Finalmente, en la parte izquierda vamos a tener el menú de archivos, donde vamos a poder crear nuevos archivos, borrarlos, etc. También tiene el acceso a otras herramientas que de momento no vamos a estar usando.

Vamos a ver que en el menú de archivos ya tenemos un archivo creado, llamado `main.py`. Este archivo es el archivo principal de nuestro programa, y es el que se ejecuta cuando presionamos el botón de **Run**.

Si bien podemos tener otros archivos, el único que se ejecuta cuando presionamos **Run** es `main.py`. Por lo tanto, es importante que nuestro programa principal o lo que nosotros queramos correr, esté en este archivo. Lo que podemos hacer, es crear otros archivos para ir guardando nuestro código y ejercicios anteriores sin necesidad de que se ejecuten cada vez que presionamos **Run**.

¡Probemos el espacio de trabajo! Vamos a escribir en el archivo `main.py` el siguiente código: `print("Hola, Mundo!")`. Luego, vamos a presionar el botón de **Run** y vamos a ver el resultado en la parte derecha de la pantalla.

¡Felicitaciones! Ya escribiste tu primer programa en Python.

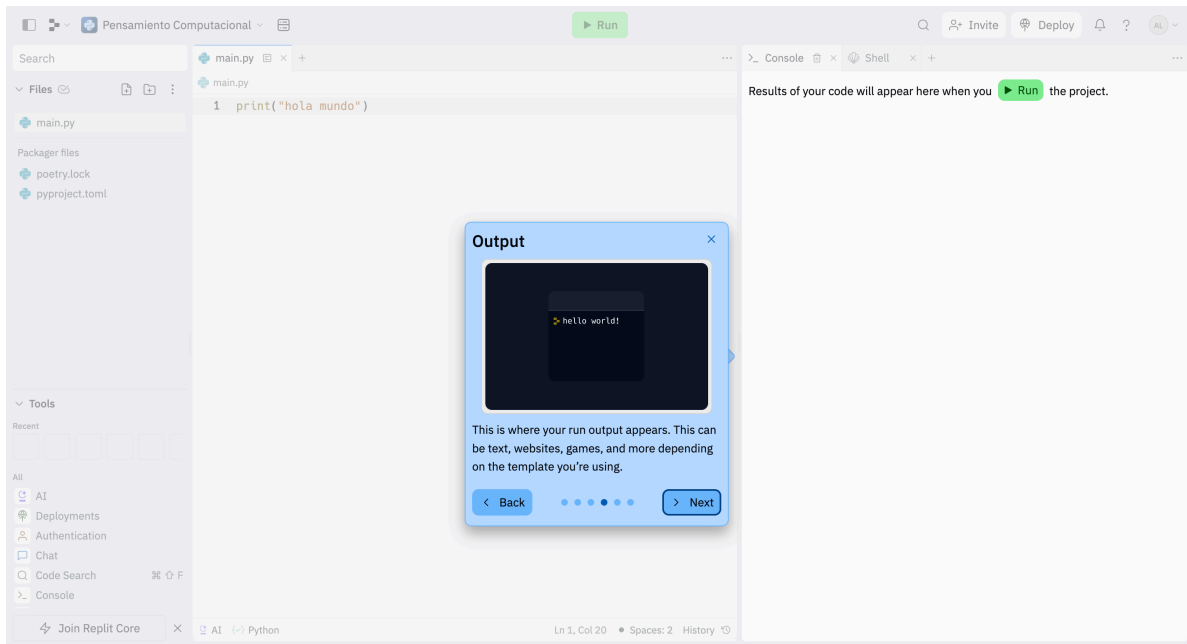


Figure 1.7: Resultado de la ejecución de código

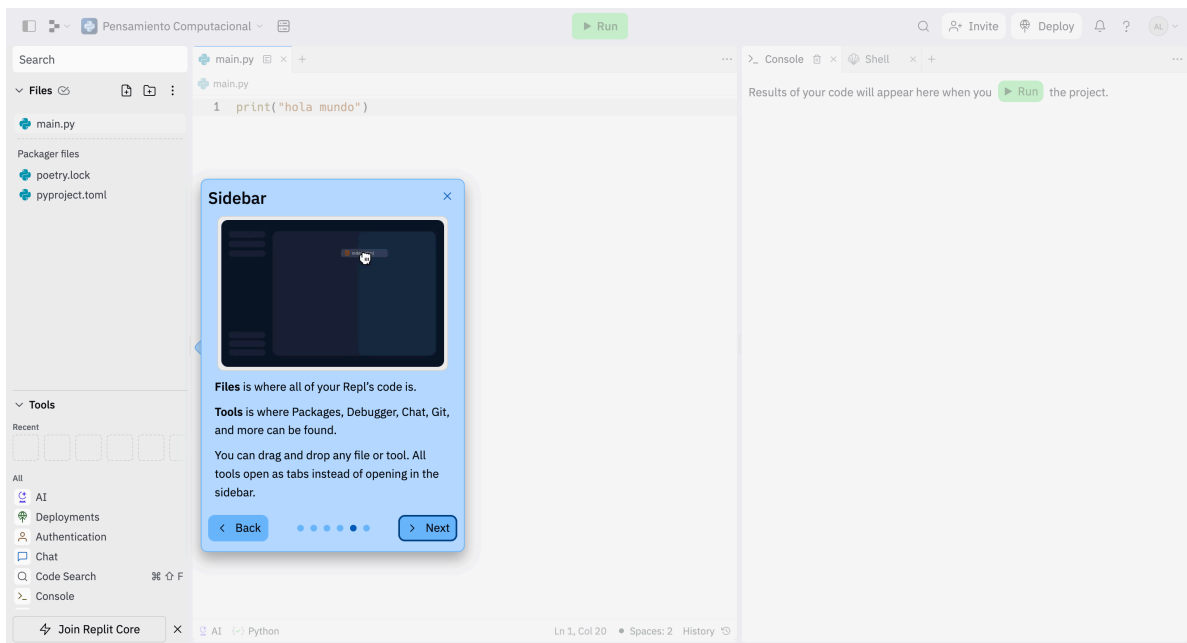


Figure 1.8: Menú de archivos



¿Lograste ver el resultado? ¿Qué pasa si presionás el botón de **Run** varias veces seguidas?

2 Tipos de Datos, Expresiones y Funciones

2.1 Sentencias Básicas

En esta unidad vamos a centrarnos en la herramienta que vamos a emplear, que es Python. Vamos a hacer un programa sencillo, interactuar con el usuario y más.

2.1.1 Flujo de Control de un Programa

El flujo de control de un programa es la forma en la que se ejecutan las instrucciones de un programa. En Python, el flujo de control es secuencial, es decir, se ejecutan las instrucciones una detrás de otra. En otros lenguajes de programación, el flujo de control puede ser condicional o repetitivo.

Ejemplo:

```
Esta línea se ejecutaría primero      ↓  
Esta línea se ejecutaría después      ↓  
Esta línea se ejecutaría a lo último
```

En este curso, la comunicación de los programas con el mundo exterior se realizará casi exclusivamente con el usuario por medio de la consola (o terminal, la presentamos en la unidad anterior en el anexo de Replit).

¡Cuidado!

Esto no significa que todos los programas siempre se comuniquen con el usuario para todo. Pensemos en las aplicaciones que usamos generalmente, como instagram: imaginémonos si para cada acción que hiciéramos dentro de la app la misma nos preguntara si queremos hacerlo o no:

- “¿Estás seguro/a de que quieres iniciar sesión?”
- “¿Estás seguro/a de que quieres traer tu nombre de usuario para mostrarse en el perfil?”
- “¿Estás seguro/a de que quieres traer tu foto de usuario para mostrarse en el perfil?”

Sería extremadamente molesto. Uno simplemente inicia sesión, y hay un montón de cosas y procesos que se ejecutan uno detrás de otro, automáticamente.

Hay cosas que no necesitan de la interacción del usuario. Nosotros nos vamos a centrar en la interacción con el usuario en gran parte del curso, pero no es lo único que se puede hacer. Los programas pueden comunicarse con otros programas y las partes de un mismo programa pueden comunicarse con otras partes del mismo programa. Más adelante vamos a ver un poco más de esta diferencia.

2.1.2 Valores y Tipos

Si tenemos la operación $7 * 5$, sabemos que el resultado es 35. Decimos que tanto 7, 5 como 35 son *valores*. En los lenguajes de programación, cada valor tiene un *tipo*.

En este caso, 7, 5 y 35 son *enteros* (o *integers* en inglés). En Python, los enteros se representan con el tipo `int`.

Python tiene dos tipos de datos numéricos: - número enteros - números de punto flotante

Los **números enteros** representan un valor entero exacto, como 42, 0, -5 o 10000.

Los **números de punto flotante** tienen una parte fraccionaria, como 3.14159, 1.0 o 0.0.

Según los operandos (los valores que se operan) y el operador (el símbolo que indica la operación), el resultado puede ser de un tipo u otro. Por ejemplo, si tenemos $7 / 5$, el resultado es 1.4, que es un número de punto flotante. Si tenemos $7 + 5$, el resultado es 12, que es un número entero.

```
1 + 2
```

```
3
```

Vamos a elegir usar enteros cada vez que necesitemos recordar, almacenar o representar un valor exacto, como pueden ser por ejemplo: la cantidad de alumnos, cuántas veces repetimos una operación, un número de documento, etc.

Vamos a elegir usar números de punto flotante cada vez que necesitemos recordar, almacenar o representar un valor aproximado, como pueden ser por ejemplo: la altura o el peso de una persona, la temperatura de un día, una distancia recorrida, etc.

```
0.1 + 0.2
```

```
0.30000000000000004
```

Como vemos, cuando hay números de punto flotante, el resultado es aproximado. $0.1 + 0.2$ nos debería dar 0.3, pero nos da 0.30000000000000004. Esto es porque los números de punto flotante son aproximados, y no pueden representar todos los valores de forma exacta. Esto es algo que vamos a tener que tener en cuenta cuando trabajemos con números de punto flotante.

Uso de punto

Notemos que para representar números de punto flotante, usamos el punto (.) y no la coma (,). Esto es porque en Python, la coma se usa para separar valores, como vamos a ver más adelante.

Además de efectuar operaciones matemáticas, Python nos permite trabajar con porciones de texto, que se llaman **cadenas** (o *strings* en inglés). Las cadenas se representan con el tipo `str`.

Las cadenas se escriben entre comillas simples (') o dobles (").

```
print( "¡Hola!" )
```

¡Hola!

```
print( '¡Hola!' )
```

¡Hola!

Las cadenas también tienen operaciones disponibles, como por ejemplo la concatenación, que es la unión de dos cadenas en una sola. Esto se hace con el operador `+`.

```
print( "¡Hola!" + " ¿Cómo estás?" )
```

¡Hola! ¿Cómo estás?

Vamos a ver más de estas operaciones más adelante.

2.1.3 Variables

Python nos permite asignarle un nombre a un valor, de forma tal que podamos “recordarlo” y usarlo más adelante. A esto se le llama **asignación**.

Estos nombres se llaman **variables**, y son espacios donde podemos almacenar valores.

La asignación se hace con el operador = de la siguiente forma: `<nombre> = <valor o expresion>`.

Ejemplos:

```
x = 5
```

```
y = x + 2
```

```
print(y)
```

7

```
print(y * 2)
```

14

```
lenguaje = "Python"
```

```
texto = "Estoy programando en " + lenguaje  
print(texto)
```

Estoy programando en Python

En este ejemplo, creamos las siguientes variables:

- x
- y
- lenguaje
- texto

y las asociamos a los valores 5, 7, “Python” y “Estoy programando en Python” respectivamente. Luego podemos usar esas variables como parte de cualquier expresión, y en el momento de evaluarla, Python reemplazará las variables por su valor asociado.

Variables y Constantes

Si el dato es inmutable (no puede cambiar) durante la ejecución del programa, se dice que ese dato es una **constante**. Si tiene la habilidad de cambiar, se dice que es una variable. En Python, todas las variables son mutables, es decir, pueden cambiar su valor durante la ejecución del programa.

Y no sólo pueden cambiar su valor, sino también su tipo: `x = 5` y `x = "Hola"` son dos asignaciones válidas, y se pueden hacer una debajo de la otra:

```
x = 5
x = "Hola"
print(x)
```

Hola

Nombres de Variables

No se puede usar el mismo nombre para dos datos diferentes a la vez; una variable puede referenciar un sólo dato por vez. Si se usa un mismo nombre para un dato diferente, se pierde la referencia al dato anterior.

2.1.4 Funciones

Para poder realizar algunas operaciones particulares, necesitamos introducir el concepto de *función*. Una función es un bloque de código que se ejecuta cuando se la llama.

Es un fragmento de programa que permite efectuar una operación determinada. `abs`, `print`, `max` son ejemplos de funciones de Python: `abs` permite calcular el valor absoluto de un número, `print` permite mostrar un valor por pantalla y `max` permite calcular el máximo entre dos valores.

Una función puede recibir 0 o más *parámetros* o *argumentos*, que son valores que se le pasan a la función entre paréntesis y separados por comas, para que los use.

```
abs(-5)
```

```
print("¡Hola!")
```

¡Hola!

```
max(5, 7)
```

7

La función recibe los parámetros, efectúa una operación y devuelve un *resultado*.

Python viene equipado de muchas funciones predefinidas, pero nosotros como programadores debemos ser capaces de escribir nuevas instrucciones para la computadora. Las grandes aplicaciones como el correo electrónico, navegación web, chat, juegos, etc. no son más que grandes programas implementados introduciendo nuevas funciones a la máquina, escritas por uno o más programadores.

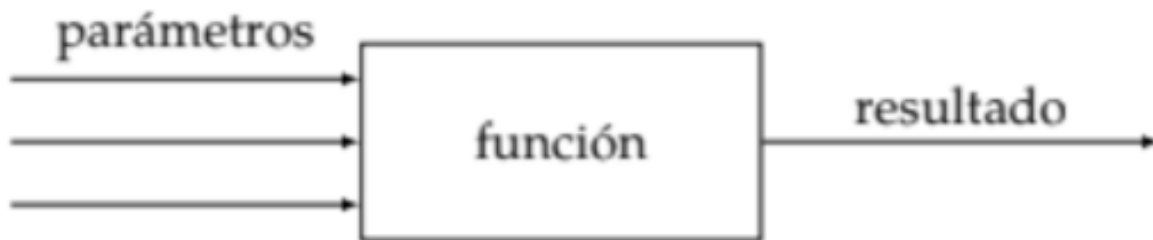


Figure 2.1: Una función recibe parámetros y devuelve un resultado

i Python es Case Sensitive

Python es Case Sensitive, es decir, distingue entre mayúsculas y minúsculas. Es muy importante respetar mayúsculas y minúsculas: `PRINT()` o `prINT()` no serán reconocidas. Esto aplica para todo lo que escribamos en nuestros programas.

Si queremos crear una función que nos devuelva un saludo a Lucia cada vez que se la llama, debemos ingresar el siguiente conjunto de líneas en Python:

```
def saludar_lucia():  
    return "Hola, Lucia!"
```

Varias cosas a notar del código:

1. `saludar_lucia` es el nombre de la función. Podría ser cualquier otro nombre, pero es una buena práctica que el nombre de la función describa lo que hace.
2. `def` es una palabra clave que indica que estamos definiendo una función.
3. `return` indica el valor que devuelve la función. Es decir, el *resultado*. Puede devolverse una sola cosa, como en este caso, o varias cosas separadas por comas.
4. La sangría (el espacio inicial) en el renglón 2 le indica a Python que estamos dentro del *cuerpo* de la función. El *cuerpo* de la función es el bloque de código que se ejecuta cuando se llama a la misma.

Sangría

La sangría puede ingresarse utilizando dos o más espacios, o presionando la tecla Tab. Es importante prestar atención en no mezclar espacios con tabs, para evitar “confundir” al intérprete.

Firma de la función

La firma de una función es la primera línea de la misma, donde se indica el nombre de la función y los parámetros que recibe. La firma permite identificar y diferenciar a una función de otra.

Pero, como vemos, el bloque de código anterior no hace nada. Para que la función haga algo, tenemos que llamarla. Para llamar a una función, escribimos su nombre, seguido de paréntesis y los parámetros que recibe, separados por comas.

```
saludar_lucia()
```

Se dice que estamos *invocando* o *llamando* a la función. Y al invocar una función, se ejecutan las instrucciones que habíamos escrito en su cuerpo.

Pero de nuevo, vemos que no pasa nada. ¿Por qué? Porque la función usa `return` para devolver un valor. Pero nosotros no estamos haciendo nada con ese valor. Para poder verlo, tenemos que imprimirlo por pantalla.

```
saludo = saludar_lucia()
print(saludo)
```

Hola, Lucia!

Lo que hicimos fue asignar el resultado devuelto por `saludar_lucia` a la variable `saludo`, y luego imprimir el valor de la variable por pantalla.

Bueno, ahora podemos saludar a Lucia. Pero vamos a querer saludar a otras personas también. ¿Cómo hacemos? Podemos hacer una función que reciba el nombre de la persona a saludar como parámetro.

```
def saludar(nombre):  
    return "Hola, " + nombre + "!"
```

De esta forma, podemos saludar a cualquier persona, pasando su nombre como parámetro.

```
# Esta es otra forma de imprimir, sin necesidad de guardarnos  
# el resultado de la función en una variable,  
# simplemente la imprimimos  
print(saludar("Lucia"))
```

Hola, Lucia!

```
print(saludar("Serena"))
```

Hola, Serena!

2.1.4.1 Ejemplos

Ejemplo

Escribir una función que calcule el doble de un número.

```
def obtener_doble(numero):  
    return numero * 2
```

Para invocarla, debemos llamarla pasándole un número:

```
doble = obtener_doble(5)  
print(doble)
```

10

Ejemplo

Pensá un número, duplícalo, súmale 6, divídelo por 2 y resta el número que elegiste al comienzo. El número que queda es siempre 3.

```
def f(numero):  
    return ((numero * 2) + 6) / 2 - numero  
  
print(f(5))
```

3.0

2.1.5 Ingreso de Datos por Consola

Hasta ahora, los programas que hicimos no interactuaban con el usuario. Pero para que nuestros programas sean más útiles, vamos a querer que el usuario pueda ingresar datos, y que el programa pueda mostrarle datos por pantalla. Para esto, vamos a usar la función `input`.

```
input()
```

`Input` es una función que bloquea el flujo del programa, esperando a que el usuario ingrese una entrada por consola y presione *enter*. Cuando el usuario presiona *enter*, la función devuelve el valor ingresado por el usuario.

```
input()  
print("terminé!")
```

Si corremos el bloque de código anterior, vamos a tener un comportamiento como este:

1. La consola va a quedar vacía, esperando el ingreso del usuario
2. Ingresamos un valor, el que tengamos ganas, y presionamos *enter*.
3. La consola muestra el mensaje “terminé!”.

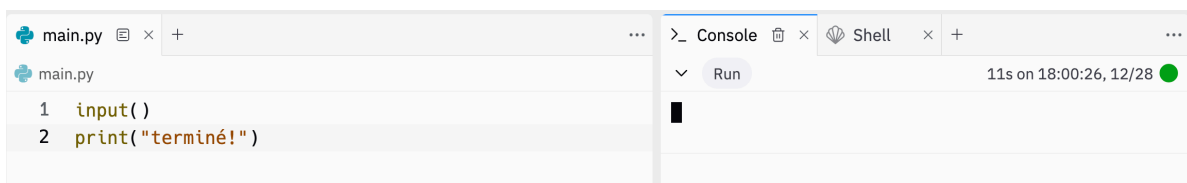
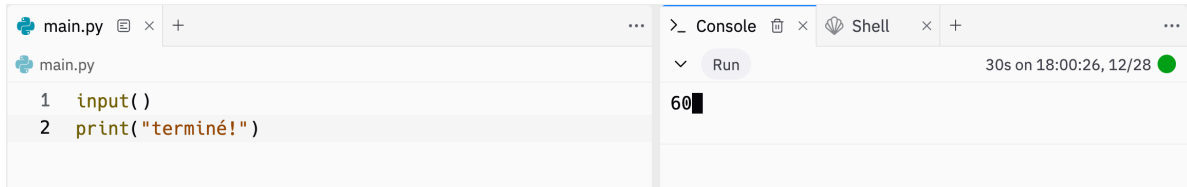


Figure 2.2: Input bloquea el flujo del programa

2.1.5.1 Obteniendo el Valor Ingresado

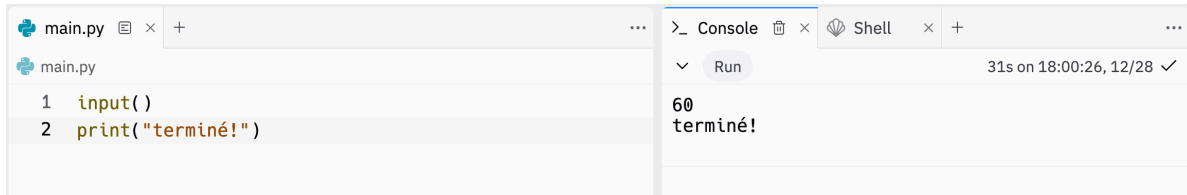
Como dijimos más arriba, la función `input` devuelve el valor ingresado por el usuario. Para poder usarlo, tenemos que guardarlo en una variable.



```
main.py 1 input()
2 print("terminé!")
```

```
>_ Console 30s on 18:00:26, 12/28
60
```

Figure 2.3: Ingresamos un valor (puede ser un número, texto, o ambos)

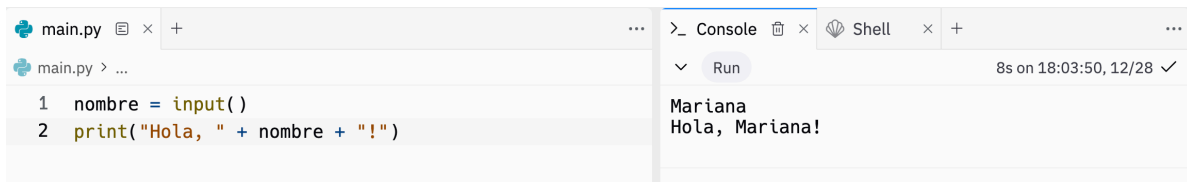


```
main.py 1 input()
2 print("terminé!")
```

```
>_ Console 31s on 18:00:26, 12/28 ✓
60
terminé!
```

Figure 2.4: Al presionar Enter, la consola muestra el mensaje “terminé!”

```
nombre = input()
print("Hola, " + nombre + "!")
```



```
main.py 1 nombre = input()
2 print("Hola, " + nombre + "!")
```

```
>_ Console 8s on 18:03:50, 12/28 ✓
Mariana
Hola, Mariana!
```

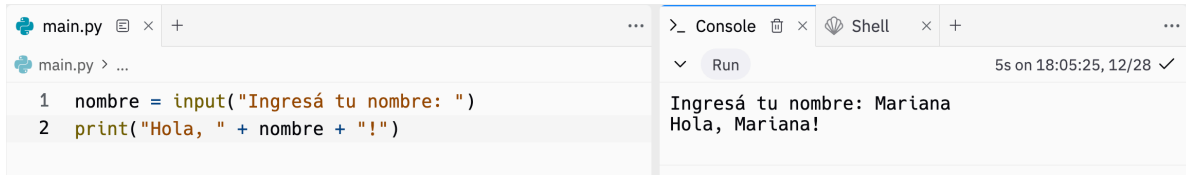
Figure 2.5: Ingresamos “Mariana” y presionamos Enter.

Para hacer nuestro programa más amigable, podemos mostrarle al usuario un mensaje antes de pedirle que ingrese un valor. Para esto, podemos pasarle un parámetro a la función `input`, que es el mensaje que queremos mostrarle al usuario.

```
nombre = input("Ingresá tu nombre: ")
print("Hola, " + nombre + "!")
```

⚠ ¡Cuidado!

A partir de la guía 2, a menos que el ejercicio diga específicamente “pedirle al usuario”, no se debe usar `input`, sino que todo tiene que recibirse por parámetro en la función. Lo mismo con `print`: A menos que el ejercicio diga específicamente “imprimir”, todo siempre se tiene que devolver con un `return`.



```
main.py 1 nombre = input("Ingresa tu nombre: ")
2 print("Hola, " + nombre + "!")

>_ Console Shell
Run 5s on 18:05:25, 12/28 ✓
Ingresa tu nombre: Mariana
Hola, Mariana!
```

Figure 2.6: Ingresamos “Mariana” y presionamos Enter.

2.2 Buenas Prácticas de programación

2.2.1 Sobre Comentarios

Los comentarios son líneas que se escriben en el código, pero que no se ejecutan. Sirven para que el programador pueda dejar notas en el código, para que se entienda mejor qué hace el programa.

Los comentarios se escriben con el símbolo `#`. Todo lo que esté a la derecha del `#` no se ejecuta. También se pueden encerrar entre tres comillas dobles (`"""`) para escribir comentarios de varias líneas.

```
# Esto es un comentario

""" Esto es un comentario
de varias líneas """
```

No es correcto escribir comentarios que no aporten nada al código, o tener el código absolutamente plagado de comentarios. Los comentarios deben ser útiles, y deben aportar información que no se pueda inferir del código. Nuestro primer intento de hacer el código más entendible no tienen que ser los comentarios, sino mejorar el código en sí.

2.2.2 Sobre Convención de Nombres

Para nombres de variables y funciones, se usa `snake_case`, que es básicamente dejar todas las palabras en minúscula y unir las con un guión bajo. Ejemplos: `numero_positivo`, `sumar_cinco`, `pedir_numero`, etc. Siempre emplear un nombre que nos remita al significado que tendrá ese dato, siempre en `snake_case`: `numero`, `letra`, `letra2`, `edad_hermano`, etc.

2.2.2.1 Variables

Las variables son cosas. Entonces sus nombres son sustantivos: `nombre`, `numero`, `suma`, `resta`, `resultado`, `respuesta_usuario`. La única excepción son las variables booleanas (ya las vamos

a ver, son aquellas que pueden guardar dos posibles valores: verdadero o falso), que suelen tener nombres como `es_par`, `es_cero`, `es_entero`, porque su valor es `true` o `false`.

A veces es útil alguna frase para identificar mejor el contenido:

`edad_mayor_hijo`, `apellido_conyuge`

2.2.2.2 Funciones

Las funciones hacen algo. Entonces sus nombres son verbos. Se usan siempre verbos en infinitivo (terminan en `-ar`, `-er`, `-ir`): `calcular_suma`, `imprimir_mensaje`, `correr_prueba`, `obtener_triplicado`, etc.

De nuevo, las excepciones son las funciones que devuelven un valor booleano (V o F). Esas pueden llamarse como: `es_par`, `da_cero`, `tiene_letra_a`, porque devuelven verdadero o falso, y eso nos confirma o niega la afirmación que hace el nombre.

2.2.3 Sobre Ordenamiento de Código

Cuando uno corre Python, lo que hace el lenguaje es leer línea a línea nuestro código. Lo que se puede ejecutar, lo ejecuta. Las funciones las guarda en memoria para poder usarlas luego. Entonces es más ordenado y prolijo primero poner todas las funciones, y después el código “ejecutable” (si van a dejar código suelto en el archivo).

Además, no olvidemos que Python tiene un flujo de control de arriba para abajo. Si intentamos invocar funciones antes de que estén definidas (`def`), Python no va a saber qué hacer, y nos va a tirar un error.

Esto es correcto:

Esto es incorrecto:

2.2.4 Sobre uso de Parámetros en Funciones

Una función se puede pensar como una caja cerrada o una fábrica. La función tiene dos puertas: una de entrada y una de salida.

La puerta de entrada son los **parámetros** y la de salida es el **output** (el resultado).

Cuando se llama o invoca a la función, la puerta de entrada se abre, permitiéndonos enviarle (pasarle) cero, uno o más parámetros a la función (según cómo esté definida). Los parámetros son datos que la función necesita para funcionar, y como ya dijimos, se le pasan a la misma entre los paréntesis de la llamada.

Ejemplo: `saludar(nombre)`, `imprimir_elementos(lista)`, `sumar(numero1, numero2)`, etc.

Una vez que la función se empieza a ejecutar, ambas puertas se cierran. Esto quiere decir que, mientras la función se está ejecutando, nada entra y nada sale de la misma.

La función debería trabajar únicamente con los datos que se le hayan pasado por parámetro o que se le pidan al usuario dentro de ella, pero no debería utilizar nada que esté por fuera de la misma.

⚠ ¡Cuidado!

Python nos deja usar cosas por fuera de la función y sin recibir los datos por parámetro, porque es un lenguaje muy benevolente. Pero está mal usar cosas que no se hayan recibido por parámetro: es una mala práctica.

Una vez que la función terminó de ejecutarse, el o los valores de salida (resultados) se devuelven por el output. Una función puede retornar uno o más elementos, o podría simplemente no retornar nada.

`return suma`, `return numero1, numero2`, `return`, etc.

Podemos ver la diferencia entre enviar algo por parámetro y usarlo por fuera de la función a continuación:

Esto está mal

Esto está bien

```
def saludar():  
    print("Hola, " + nombre + "!")
```

```
nombre = "Manuela"  
saludar()
```

```
def saludar(persona):  
    print("Hola, " + persona + "!")
```

```
nombre = "Manuela"  
saludar(nombre)
```

💡 Tip

Como podemos observar los nombres de los argumentos cuando se invoca y en la definición de la firma pueden ser los mismos o distintos. En este caso, la función sabe

que está recibiendo algo como parámetro, y sabe que dentro de su cuerpo a este dato lo va a identificar como **persona**, pero no hace falta que la variable que nosotros le pasamos como parámetro también se llame **persona**: en este caso se llama **nombre**.

2.3 Tipos de Datos

2.3.1 Datos Simples

Los programas trabajan con una gran variedad de datos. Los datos más simples son los que ya vimos: números enteros, números de punto flotante y cadenas.

Pero dependiendo de la naturaleza o el **tipo** de información, cabrá la posibilidad de realizar distintas transformaciones aplicando **operadores**. Por eso, a la hora de representar información no sólo es importante que identifiquemos al dato y podamos conocer su valor, sino saber qué tipo de tratamiento podemos darle.

Todos los lenguajes tienen tipos predefinidos de datos. Se llaman predefinidos porque el lenguaje ya los conoce: sabe cómo guardarlos en memoria y qué transformaciones puede aplicarles.

En Python, tenemos los siguientes tipos de datos:

Tipo	Descripción	Ejemplo
<code>int</code>	Números enteros	5, 0, -5, 10000
<code>float</code>	Números de punto flotante o reales	3.14159, 1.0, 0.0
<code>complex</code>	Números complejos	(1, 2j), (1.0,-2.0j), (0,1j). La componente con j es la parte imaginaria.
<code>bool</code>	Valores booleanos o valores lógicos	True, False
<code>str</code>	Cadenas de caracteres	"Hola", "Python", "¡Hola, mundo!", "" (cadena vacía, no contiene ningún caracter)

i ¿Por qué se llaman “cadenas de caracteres”?

Porque son una cadena de caracteres, es decir, una secuencia de caracteres. Por ejemplo, la cadena “Hola” está formada por los caracteres “H”, “o”, “l” y “a”. Esto nos permite acceder a cada uno de los caracteres de la cadena por separado si quisiéramos, o a

porciones de una cadena, como vamos a ver más adelante.

Más aún, podemos ver que el texto “hola” no será igual a “aloh” ni a “Holá”, porque son cadenas distintas.

Un string permite almacenar cualquier tipo de caracter unicode dentro (letras, números, símbolos, emojis, etc.).

2.3.2 Operadores Numéricos

Los operadores son símbolos que representan una operación. Por ejemplo, el operador + representa la suma.

Para transformar datos numéricos, emplearemos los siguientes operadores:

Símbolo	Definición	Ejemplo
+	Suma	5 + 3
-	Resta	5 - 3
*	Producto	5 * 3
**	Potencia	5 ** 2
/	División	5 / 3
//	División entera	5 // 3
%	Módulo o Resto	5 % 3
+=	Suma abreviada	x = 0x += 3
-=	Resta abreviada	x = 0x -= 3
*=	Producto abreviado	x = 0x *= 3
/=	División abreviada	x = 0x /= 3
//=	División entera abreviada	x = 0x //= 3
%=	Módulo o Resto abreviado	x = 0x %= 3

Como pasa en matemática, para alterar cualquier precedencia (prioridad de operadores) se pueden usar paréntesis.

```
(5 + 3) * 2
```

16

```
5 + (3 * 2)
```

11

El orden de prioridad de ejecución para los operadores va a ser el mismo que en matemática.

2.3.3 Operadores de Texto

Para transformar datos de texto, emplearemos los siguientes operadores:

Símbolo	Definición	Ejemplo
+	Concatenación	"Hola" + " " + "Mundo"
*	Repetición	"Hola" * 3
+=	Concatenación abreviada	x = "Hola"x += " Mundo"
*=	Repetición abreviada	x = "Hola"x *= 3
[k] o [-k]	Acceso a un caracter	"Hola"[0] "Hola"[-1]
[k1:k2]	Acceso a una porción	"Hola"[0:2] "Hola"[1:] "Hola"[:2] "Hola"[:]

De nuevo, para alterar precedencias, se deben usar ().

2.3.3.1 Manipulando Strings

Si bien esto se va a ahondar en la siguiente sesión de la materia, es importante saber que los strings, como se dijo más arriba, son un conjunto de caracteres. Pero no sólo un conjunto, sino un **conjunto ordenado**. Esto quiere decir que cada caracter tiene una posición dentro de la cadena, y que esa posición es importante.

Por ejemplo, la cadena "Hola" tiene 4 caracteres: "H", "o", "l" y "a".

La posición de cada caracter es la siguiente:

Posición	0	1	2	3
Caracter	"H"	"o"	"l"	"a"

Entonces, si queremos acceder al caracter "H", tenemos que usar la posición 0. Si queremos acceder al caracter "a", tenemos que usar la posición 3.

Tip

Para acceder a un caracter de una cadena, usamos los corchetes ([]) y dentro de ellos la posición del caracter que queremos acceder.

```
letra = "Hola"[0]
print(letra)
```

H

Pero no sólo puedo obtener los caracteres en las posiciones de la palabra, sino que puedo obtener *slices* o *porciones* de la misma, usando algo que vemos por primera vez: los **rangos**.

Un rango tiene tres partes:

```
[start : end : step]
```

- **start** es el índice de inicio del rango. Si no se especifica, se toma el índice 0. El carácter en la posición de inicio siempre se incluye.
- **end** es el índice de fin del rango. Si no se especifica, se toma el índice final de la cadena. El carácter en la posición de fin nunca se incluye.
- **step** es el tamaño del paso. Si no se especifica, se toma el valor 1.

Ejemplos:

2.3.4 Input y Casteo

Cuando usamos la función `input`, el valor que devuelve es siempre una cadena. Esto es porque el usuario puede ingresar cualquier cosa, y no sabemos qué tipo de dato es.

Por ejemplo, si le pedimos al usuario que ingrese un número, el usuario puede ingresar un número entero, un número de punto flotante, un número complejo, o incluso un texto. Entonces, el valor que devuelve `input` es siempre una cadena, y nosotros tenemos que transformarla al tipo de dato que necesitamos.

Por ejemplo:

```
edad = input("Indique su edad:")
print("Su edad es:", edad_nueva)
```

💡 Imprimiendo Strings y Variables

Existen muchas formas de concatenar variables con texto.

1. Usando el operador `+`: "Su edad es: " + edad

2. Usando el método `fstring`: `f"Su edad es: {edad}"`
3. Usando el caracter `,`: `print("Su edad es:", edad)`

La forma más recomendada es la segunda, usando `fstring`. Pero dependerá de cada caso.

El problema es que, si bien nuestro código anterior funciona, no podemos operar `edad` como si fuese un número, porque es un string.

El siguiente código va a fallar:

```
edad = input("Indique su edad:")
edad_nueva = edad + 1
print("Edad siguiente:", edad_nueva)
```

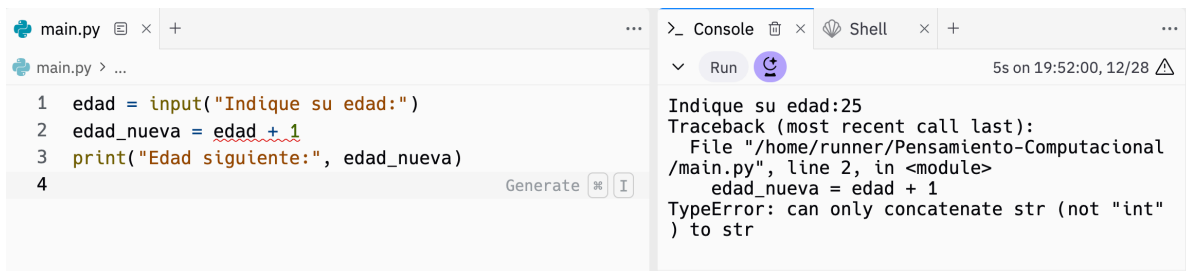


Figure 2.7: Ejecución del bloque de código

Como vemos, la consola nos arroja un error, o en términos simples decimos que “explotó”.

💡 ¿Qué es un error?

Los errores son información que nos da la consola para que podamos corregir nuestro código.

En este caso, nos dice que no se puede concatenar un string con un int.

¿Por qué nos dice eso? Porque `edad` es un string: `"25"`, y estamos tratando de sumarle 1, que es un int: `1`.

Para poder operar con `edad` como si fuese un número, tenemos que transformarla a un número. Esto se llama **castear**.

Para castear un valor a un tipo de dato, usamos el nombre del tipo de dato, seguido de paréntesis y el valor que queremos castear.

```
int("25")
```

De esta forma, podemos modificar nuestro código anterior:

```
edad = int(input("Indique su edad:")) # Le agregamos int
edad_nueva = edad + 1
print("Edad siguiente:", edad_nueva)
```

Y obtenemos un código que funciona correctamente.

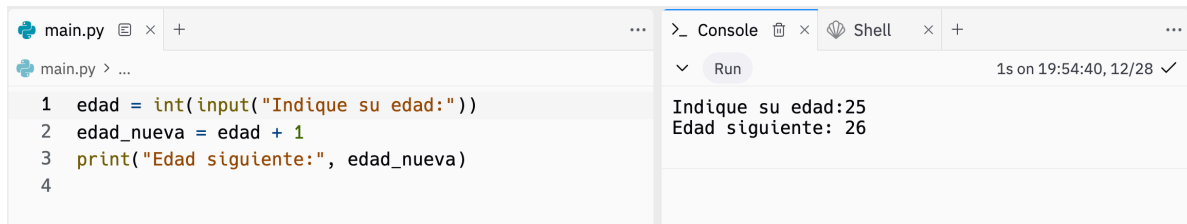


Figure 2.8: Ejecución del bloque de código

De esta forma, podemos castear a varios tipos de datos:

```
numero_entero = int(input("Ingrese un número"))
punto_flotante = float(input("Ingrese un número"))

punto_flotante2 = float(numero_entero)

numero_en_str = str(numero_entero)
```

Ejemplo:

```
nombre_menor = input('Ingresá el nombre de un conocido/a:')
edad_menor = int(input(f'Ingresá la edad de { nombre_menor } '))
nombre_mayor = input(f'Cómo se llama el hermano/a mayor de {nombre_menor}? ')
diferencia = int(input(f'Cuántos años más grande es {nombre_mayor}? '))

edad_mayor = edad_menor + diferencia

print(nombre_menor,'tiene',edad_menor,'años')
print(nombre_mayor,'es mayor y tiene', edad_mayor, 'años')
```

```

main.py
1 nombre_menor = input('Ingresá el nombre de un conocido/a:')
2 edad_menor = int(input(f'Ingresá la edad de { nombre_menor } '))
3 nombre_mayor = input(f'Cómo se llama el hermano/a mayor de {nombre_menor}? ')
4 diferencia = int(input(f'Cuántos años más grande es {nombre_mayor}? '))
5
6 edad_mayor = edad_menor + diferencia
7
8 print(nombre_menor, 'tiene', edad_menor, 'años')
9 print(nombre_mayor, 'es mayor y tiene', edad_mayor, 'años')

```

```

Run
45s on 20:06:06, 12/28 ✓

Ingresá el nombre de un conocido/a:Julieta
Ingresá la edad de Julieta 25
Cómo se llama el hermano/a mayor de Julieta? Camila
Cuántos años más grande es Camila? 7
Julieta tiene 25 años
Camila es mayor y tiene 32 años

```

Figure 2.9: Ejecución del bloque de código

2.4 Bonus Track: Algunas Funciones Predefinidas de Python

Recomendación

Te recomendamos que te animes a probar estas funciones, para ver qué hacen y terminar de entenderlas.

Función	Definición	Ejemplo de uso
<code>print()</code>	Imprime un mensaje o valor en la consola	<code>print("Hello, world!")</code>
<code>input()</code>	Lee una entrada de texto desde el usuario	<code>name = input("Enter your name: ")</code>
<code>abs()</code>	Devuelve el valor absoluto de un número	<code>abs(-5)</code>
<code>round()</code>	Redondea un número al entero más cercano	<code>round(3.7)</code>
<code>int()</code>	Convierte un valor en un entero	<code>x = int("5")</code>
<code>float()</code>	Convierte un valor en un número de punto flotante	<code>y = float("3.14")</code>
<code>str()</code>	Convierte un valor en una cadena de texto	<code>message = str(42)</code>
<code>bool()</code>	Convierte un valor en un booleano	<code>is_valid = bool(1)</code>
<code>len()</code>	Devuelve la longitud (número de elementos) de un objeto	<code>length = len("Hello")</code>

Función	Definición	Ejemplo de uso
<code>max()</code>	Devuelve el valor máximo entre varios elementos o una secuencia	<code>max(4, 9, 2)</code>
<code>min()</code>	Devuelve el valor mínimo entre varios elementos o una secuencia	<code>min(4, 9, 2)</code>
<code>pow()</code>	Calcula la potencia de un número	<code>result = pow(2, 3)</code>
<code>range()</code>	Genera una secuencia de números	<code>numbers = range(1, 5)</code>
<code>type()</code>	Devuelve el tipo de un objeto	<code>data_type = type("Hello")</code>
<code>round()</code>	Redondea un número a un número de decimales específico	<code>rounded_num = round(3.14159, 2)</code>
<code>isinstance()</code>	Verifica si un objeto es una instancia de una clase específica	<code>is_instance = isinstance(5, int)</code>
<code>replace()</code>	Reemplaza todas las apariciones de un substring por otro	<code>text = "Hello, World!" new_text = text.replace("Hello", "Hi")</code>
<code>eval(<expr>)</code>	Evalúa una expresión	<code>eval("2 + 2")</code>

3 Estructuras de Control

3.1 Subtitle

4 Tipos de Datos Estructurados

4.1 Subtitle

5 Entrada y Salida de Información

5.1 Subtitle

6 Bibliotecas

6.1 Subtitle

Referencias

Guía de Ejercicios

Guía 1: Introducción a la Algoritmia y la Programación

Recomendación

En esta guía nos dedicaremos a introducirnos en los conceptos de programación y algoritmo. Para los primeros seis ejercicios, te recomendamos ver [este video](#) para recordar cómo entiende la computadora nuestras instrucciones.

1. Se tiene que explicar a una máquina exactamente cómo servir un vaso de jugo (de los que vienen en cartón) de la heladera. Recordando la definición de algoritmo, hacer una descripción paso a paso de lo que se tiene que hacer y usar para lograr el objetivo. Pista: No vas a necesitar nada de código en este ejercicio, sólo nombrar los pasos.
2. Se tiene que explicar a una máquina exactamente cómo hacer una tostada con queso, pensá qué ingredientes se necesitan con sus cantidades, cómo tiene que ser el espacio de trabajo y los elementos que va a necesitar usar. Recordando la definición de algoritmo, hacer una descripción paso a paso de lo que se tiene que hacer y usar para hacer una tostada con queso. Pista: No vas a necesitar nada de código en este ejercicio, sólo nombrar los pasos.
3. Se te pide que organices una colecta de alimentos no perecederos por la Ciudad de Buenos Aires. Contamos con algunos automóviles y camionetas de voluntarios, un listado de donaciones, listado de los alimentos a donar, la disponibilidad horaria y la dirección en la cual se dejan los alimentos. La colecta se realiza en un solo día. ¿Cómo la organizarías? Pista: No vas a necesitar nada de código en este ejercicio, sólo nombrar los pasos.
4. Tenés que enviar invitaciones personalizadas para tu cumpleaños. Cada invitación tiene que mencionar el nombre de la persona y la relación que tiene con vos. Contamos con una impresora a la que le das el texto a enviar, un listado con los nombres de los invitados y la relación que cada uno tiene con vos. ¿Cómo redactarías el texto de la invitación? Pista: No vas a necesitar nada de código en este ejercicio, sólo nombrar los pasos.
5. Se te encargó definir qué datos son necesarios para el registro de estudiantes en un curso de inglés. ¿Qué datos crees que deberían ser obligatorios y cuáles opcionales? ¿Y si el curso es de cocina? Pista: No vas a necesitar nada de código en este ejercicio, sólo nombrar los pasos.

6. Contás con un listado de cosas a comprar y tenes que ir a un supermercado que cuenta con distintas góndolas o pasillos. Cada góndola o pasillo puede contar con varios, uno o ninguno de los productos de tu lista. ¿Cuál sería el listado de instrucciones para poder terminar lo más rápido posible? Pista: No vas a necesitar nada de código en este ejercicio, sólo nombrar los pasos.
7. Con el anexo de Replit de la Unidad 1, realizá tu primer programa: hacé que se imprima por pantalla un "¡Hola mundo!".

Guía 2: Tipos de Datos, Expresiones y Funciones

Variables y Tipos de Dato

1. Guardar el texto "Hola, Mundo!" en una variable e imprimirla por pantalla.
2. Guardar los números 1, 2 y 3 en tres variables distintas e imprimirlos por pantalla.
3.
 - a. Guardar los números 1, 2 y 3 en tres variables distintas y luego sumarlos e imprimir el resultado por pantalla.
 - b. Repetir con las distintas operaciones disponibles que se vieron en la unidad 2: resta, multiplicación, división, división entera, resto, potencia; combinando los números entre sí.
4. Crear un programa que le solicite al usuario:
 - a. Su nombre y lo imprima por pantalla.
 - b. Su edad y la imprima por pantalla.
 - c. Su edad, le sume 1, y la imprima por pantalla.
5. Crear un programa que le solicite al usuario un número, y que determine si es par mostrando en pantalla un los mensajes "par" o "impar" según corresponda.
Para determinar si un número es par o impar, podés ver qué pasa cuando lo dividís por 2. Si el resto es 0, es par, si el resto es 1, es impar. ¿Qué operador vimos para obtener el resto?
6. Escribir un programa que le pida al usuario su año de nacimiento, y que le diga qué edad tiene en el año actual.
7. Crear un programa que le solicite al usuario 5 enteros y que muestre por pantalla el promedio de ellos. Hacerlo de dos formas:
 - a. Primero, usando 5 variables para cada entero.
 - b. Después, usando una sola variable para almacenar la suma de los 5 enteros. ¿Cómo se te ocurre que podrías hacer?

8. Crear una **función** que reciba un número y que devuelva el valor absoluto.
9. Crear una **función** que reciba un número y que devuelva **True** si es par, y **False** si es impar.
10. Crear una **función** que reciba un número y un string, y que devuelva ambos concatenados dentro de un nuevo string.
11. Crear una **función** que reciba dos enteros y que devuelva el resto y el cociente entre ellos.
12. Crear una función que le pida al usuario su nombre y apellido, e los imprima con el siguiente formato: “Apellido, Nombre”.
13. Hacer una **función** que reciba una palabra y devuelva la cantidad de letras que tiene.
14.
 - a. Hacer una **función** que reciba una palabra y que imprima los primeros 5 caracteres únicamente. Ejemplo: Si se recibe “pensamiento” se debe imprimir “pensa”.
 - b. Hacer una **función** que reciba una palabra y que imprima sólo los caracteres ubicados en posiciones pares. Ejemplo: Si se recibe “pensamiento” se debe imprimir “pnaino”.
 - c. Hacer una **función** que reciba una palabra y que imprima la palabra dada vuelta. Ejemplo: Si se recibe “materia” se debe imprimir “airectam”.
15. Hacer una **funcion** que reciba una palabra, le borre todas las letras “a” e imprima el resultado por pantalla. Pista: usar una función predefinida de Python. Ejemplo: Si se recibe “casa” se debe imprimir “cs”.
16. Analizar qué tipo de dato (o error) se obtiene al hacer las siguientes operaciones:
 - a. `5 / 2`
 - b. `5 // 2`
 - c. `5 % 2`
 - d. `5 ** 2`
 - e. `5.0 / 2`
 - f. `5.0 // 2`
 - g. `5.0 % 2`
 - h. `5.0 ** 2`
 - i. `5 / 2.0`
 - j. `5 // 2.0`
 - k. `5 % 2.0`
 - l. `5 ** 2.0`
 - m. `5.0 / 2.0`
 - n. `5.0 // 2.0`
 - o. `5.0 % 2.0`
 - p. `5.0 ** 2.0`
 - q. `"Hola" * 2`

```
r. "Hola" + 2  
s. "Hola" + "2"  
t. x = "Hola"  
   x += " mundo"
```