FUNDAMENTOS DE ALGORITMOS

MÓDULO 1

RECURSO DE APRENDIZAJE 2

INTRODUCCIÓN A PYTHON

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Escuela de Ingeniería Informática

Python

- **Python** es un **Lenguaje de programación** de **alto nivel, interpretado** que fue creado a principios de los 90 por **Guido Van Rossum.**

- Este lenguaje es muy **poderoso**, **flexible**, **sencillo** y **fácil** de aprender.

- Puede ser usado, estudiado, copiado, modificado y redistribuido **libremente.**

Principales Características de Python

Python es un lenguaje :

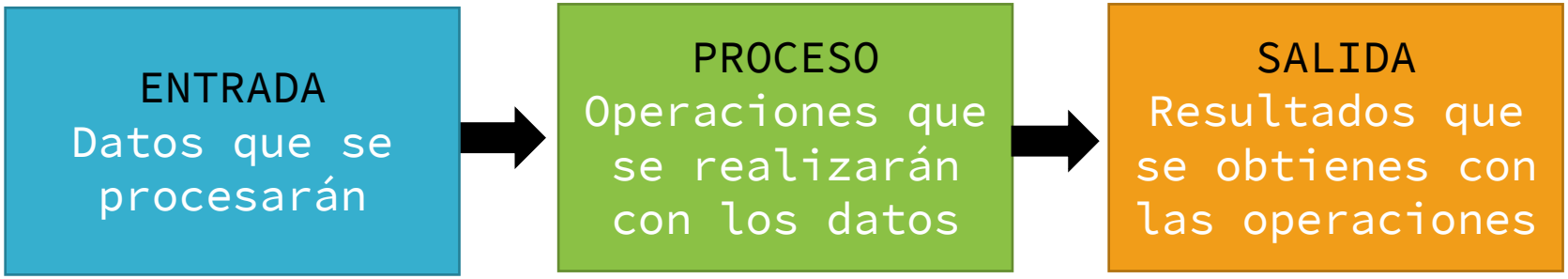
- **De Propósito General**: esto significa que permite crear todo tipo de programas o aplicaciones.
- **Multiplataforma**: esto significa que un programa en **Python** puede ser ejecutado en diferentes sistemas operativos y plataformas.
- **Multiparadigma**: esto significa que **Python** soporta varios **paradigmas de programación** (programación estructurada, programación orientada al objeto, programación imperativa y en menor medida programación funcional).

ELEMENTOS BÁSICOS PARA PROGRAMAR EN PYTHON

Programas en Python

- En este **curso** aprenderemos a escribir **programas en Python**, para resolver **problemas de procesamiento de datos**.

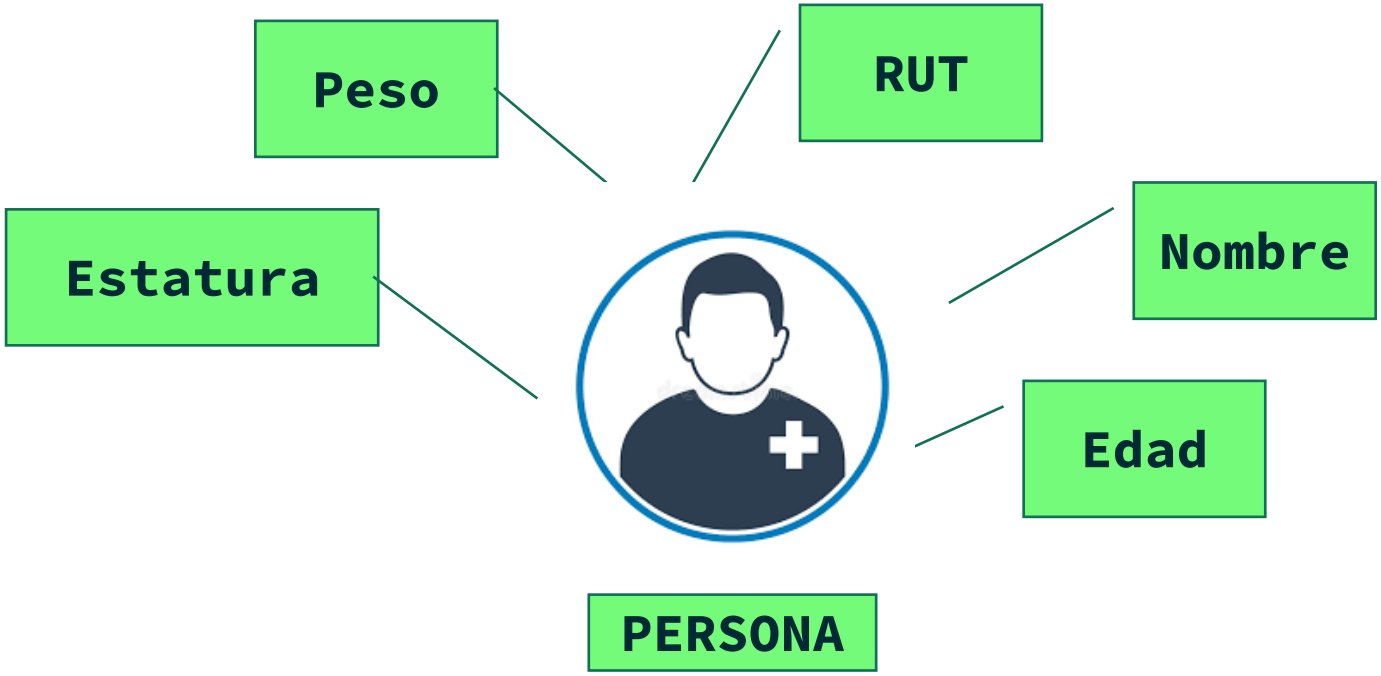
- Estos **programas** leerán **datos de entrada** que serán **procesados** para obtener los **resultados o datos de salida** esperados.



DATO Y TIPOS DE DATOS EN PYTHON

Qué es un Dato ?

- Un **dato** es una **representación simbólica** de un **atributo** o **variable cuantitativa** o **cualitativa** de un **objeto/entidad** del mundo real.



Tipo de Dato

- Un **dato** puede ser un **texto**, un valor **numérico entero**, un valor **numérico real**, etc.

Por ejemplo :

- El **peso** de una persona en **kilogramos** es un **dato de tipo numérico real**.
- La **edad** de una persona en **años** es un **dato de tipo numérico entero**.
- El **nombre** de una persona es un **dato** de tipo **cadena de caracteres o texto**.

Tipos de Datos Básicos en Python

El lenguaje Python incluye **3 tipos de datos básicos** :

1. El tipo **numérico**
2. El tipo **cadena de caracteres** (string)
3. El tipo **booleano**

Tipos Numéricos en Python

Los tipos de datos **numéricos en Python** permiten almacenar **números** en diferentes formatos :

1. entero (**int**) → ejemplos : -58 , 567 , 1234
2. Real (**float**) → ejemplos : 10.567, 10.65 , -3.45
3. Complejo (**complex**) → ejemplos : 2 + 1J , 3 + 2j

TIPO	DESCRIPCIÓN	RANGO DE VALORES QUE PUEDE ALMACENAR UN DATO DE ESTE TIPO
int	Número entero con precisión fija	Desde -9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807 (64 bits)
float	Coma flotante de doble precisión	Desde ±2,2250738585072020 x 10-308 hasta ±1,7976931348623157×10308 (64 bits)
complex	Parte real y parte imaginaria j	Ejemplo : 2 + 1j

Tipo Cadena de Caracteres en Python

- En **Python** el tipo de dato **cadena** es una **secuencia de caracteres** (numéricos, alfabéticos, espacios, signos de puntuación, etc.)

- Estos **caracteres** son los que podemos ver en nuestro **teclado** y algunos que podemos generar usando una **combinación de teclas.**

- Una **cadena** debe ir **encerrada** entre **comillas simples** o **dobles,** como se muestra en los siguientes ejemplos :

  - ‘Esta es una cadena’
  - "HOLA MUNDO"
  - ‘123456’
  - ‘AAAA@#$$////???’

Tipo Booleano en Python

- Un **dato** de tipo **booleano en Python** sólo puede almacenar dos valores:
  - **True** (verdadero)
  - **False** (Falso)
- En el **contexto** de las **operaciones booleanas**, y también cuando las **expresiones** son usadas en sentencias de **control de flujo**, los siguientes valores son interpretados como **FALSO** :
  - **False**
  - **0 (cero)**

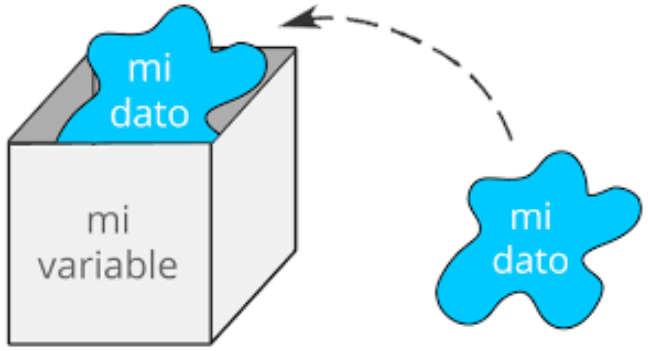
VARIABLES

Variables

- Los **datos** de **entrada**, **proceso** y **salida** de nuestros **programas** los almacenaremos en **variables**.

- Una **variable** es un **objeto** en la **memoria** del computador que tiene un **nombre asociado** y **almacena datos**.

- El **dato** que almacena una variable puede ser modificado todas las veces que sea necesario en nuestros programas.



Variables en Python

- A diferencia de otros lenguajes de programación en **Python las variables NO se declaran.**

- Pero para poder usar una variable en Python **debemos asignarle un valor inicial.**

- El **valor inicial** que le damos a nuestra **variable determina su tipo** y las **operaciones** que podemos realizar sobre ella.

Cómo asignamos valores a nuestras variables en Python ?

- Para asignarle **datos** a nuestras **variables en Python** debemos utilizar el **operador de asignación** que es el signo **=** (igual)**.**

- La **sintaxis** para asignar un dato a una **variable** es :

nombreVariable = dato

- Esta **instrucción** permite **almacenar** el **dato** de la derecha de la expresión en la **variable** que se encuentra a la izquierda de la expresión.

Ejemplos

Los siguientes ejemplos nos muestran como podemos asignar **valores/datos** de diferente **tipo** a algunas **variables** :

```
nombreEstudiante = "Ignacia Urrutia"

rutEstudiante = "22222222-2"

edadEstudiante = 18

pesoEstudiante = 55.50

estaturaEstudiante = 1.65

estudianteEsSoltera = True
```

Reglas para darle nombre a nuestras Variables

Los **nombres** (identificadores) que podemos darle a nuestras **variables** en **Python** deben **respetar** ciertas **reglas**.

1. Los nombres solo pueden incluir una combinación de letras en **minúsculas** o **MAYÚSCULAS** del **alfabeto inglés** o **dígitos** del 0..9 o el símbolo _ (**underscore**).

2. El primer carácter del nombre **NO puede ser un dígito**.

3. **NO se pueden usar** como nombres las **palabras reservadas** del lenguaje.

Palabras Reservadas

- En un lenguaje de programación, una **palabra reservada o clave** es una **palabra** que tiene un significado **gramatical** especial para dicho lenguaje.

- Estas **palabras reservadas** no pueden ser utilizadas como un **identificador/nombre** de **objetos** tales como variables, funciones etc.

Palabras Reservadas en Python

- **Python** dispone de un conjunto de **palabras reservadas** que forman su núcleo y que permiten realizar diferentes **operaciones.**

- Las **palabras reservadas de Python** se muestran en la siguiente tabla y deben ser escritas exactamente como aparecen en ella :

False	**assert**	**continue**	**except**	**if**	**nonlocal**	**return**
None	**async**	**def**	**finally**	**import**	**not**	**try**
True	**await**	**del**	**for**	**in**	**or**	**while**
and	**break**	**elif**	**from**	**is**	**pass**	**with**
as	**class**	**else**	**global**	**lambda**	**raise**	**yield**

Ejemplos de Nombres de Variables Válidos y NO válidos en Python

Ejemplos de nombres de variables válidos :

- **Nota_Tarea_1**
- **notaTarea1**
- **NotaTarea1**

Ejemplos de nombres de variables NO válidos :

- **1_Nota** (ya que parte con un dígito)
- **Edad-estudiante** (ya que usa el carácter guión - que no esta permitido)
- **False** (ya que es una palabra reservada de Python)

Buenas Prácticas al Nombrar Variables

1. Use nombres de variables **significativos** que documenten el dato que estas almacenarán.

    - **Por ejemplo**, si desea almacenar el promedio de notas de un estudiante llame a su variable **promedioNotas o promedio** en vez de **x**.

2. Seleccione y use una **convención** para **nombrar variables con identificadores compuestos (que tienen más de una palabra)** de manera **consistente**. Las convenciones más usadas son :

    - **snake_case :** edad_estudiante
    - **camelCase :** edadEstudiante
    - **CapWords :** EdadEstudiante

OPERADORES EN PYTHON

Operadores en Python

- **Python** incluye un conjunto de **operadores** para realizar **operaciones** sobre sus **distintos tipos de datos**.

Los operadores que incluye son :

1. **Operadores aritméticos.**
2. **Operadores relacionales o de comparación.**
3. **Operadores booleanos.**
4. **Operadores para cadenas de caracteres.**

OPERADORES ARITMÉTICOS

Operadores Aritméticos

- Los **operadores aritméticos en Python** nos permiten realizar **operaciones** sobre **datos** de **tipo numérico (entero, real, complejo).** La siguiente imagen nos muestra los operadores aritméticos que tiene **Python:**

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	**	Binario	Por la derecha	1
Identidad	+	Unario	—	2
Cambio de signo	-	Unario	—	2
Multiplicación	*	Binario	Por la izquierda	3
División	/	Binario	Por la izquierda	3
División entera	//	Binario	Por la izquierda	3
Módulo (o resto)	%	Binario	Por la izquierda	3
Suma	+	Binario	Por la izquierda	4
Resta	-	Binario	Por la izquierda	4

Tabla 2.1: Operadores para expresiones aritméticas. El nivel de precedencia 1 es el de mayor prioridad y el 4 el de menor.

Ejemplos Uso Operadores Aritméticos

Operador	Expresión Aritmética	Resultado almacenado en variable
Exponenciación	`variable = 2 ** 3`	8
Cambio de signo	`variable = -3`	-3
Multiplicación	`variable = 2 * 3`	6
División	`variable = 5 / 2`	2.5
División entera	`variable = 5 // 2`	2
Módulo o resto de la división	`variable = 5 % 2`	1
Suma	`variable = 5 + 2`	7
Resta	`variable = 5 - 2`	3

Ejemplos Uso Operadores Aritméticos

- Podemos formular **expresiones matemáticas** que hagan uso de **diferentes operadores aritméticos** para realizar un cálculo específico. Por ejemplo, si necesitamos calcular el promedio de 3 notas obtenidas por un estudiante en un determinado curso, podríamos escribir las siguientes instrucciones :

```
nota1 = 5.5
nota2 = 5.0
nota3 = 6.5
promedioNotas = (nota1 + nota2 + nota3) / 3
```

- Para calcular **correctamente** el promedio de notas, debemos considerar que el **operador suma** tiene **menor precedencia** que el **operador división**. Por lo tanto, necesitaremos usar los **paréntesis redondos** para **forzar prioridad,** es decir para que la suma de notas se calcule primero y luego el resultado obtenido se divida por 3.

OPERADORES RELACIONALES

Operadores Relacionales

- Los **operadores relacionales en Python** nos permiten comparar dos valores del **mismo tipo**. El **resultado** de esta **comparación** es siempre un **valor de verdad (True o False)**.

- La siguiente imagen nos muestra los operadores relacionales que tiene **Python:**

operador	comparación
!=	es distinto de
==	es igual que
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

Ejemplos Uso Operadores Relacionales

Operador	Variable 1	Variable 2	Expresión Relacional	Resultado
Distinto de	X = 5	Y = 10	X != Y	True
Es igual a	X = 5	Y = 10	X == Y	False
Es menor que	X = 5	Y = 10	X < Y	True
Es menor o igual que	X = 5	Y = 10	X <= Y	True
Es mayor que	X = 5.5	Y = 10.0	X > Y	False
Es mayor o igual que	X = ‘A’	Y = ‘B’	X >= Y	False

OPERADORES LÓGICOS

Operadores Booleanos o Lógicos

- Los **operadores booleanos/lógicos en Python** nos permiten realizar **operaciones** sobre **datos** o **expresiones** de **tipo booleano.**
- La siguiente imagen nos muestra los operadores lógicos que tiene **Python:**

Operación	Operador	Aridad	Asociatividad	Precedencia
Negación	**not**	Unario	—	alta
Conjunción	**and**	Binario	Por la izquierda	media
Disyunción	**or**	Binario	Por la izquierda	baja

Tabla 2.2: Aridad, asociatividad y precedencia de los operadores lógicos.

Tablas de Verdad

- Las tablas de verdad nos muestran el **valor de verdad resultante** de una expresión lógica, para cada combinación de **valores de verdad** posible.

- A continuación veremos :

  - La tabla de verdad del **and**
  - La tabla de verdad del **or**
  - La tabla de verdad del **not**

Tabla de Verdad - and

P	Q	Expresión Lógica	Resultado
True	True	P and Q	True
True	False	P and Q	False
False	True	P and Q	False
False	False	P and Q	False

Tabla de Verdad - or

P	Q	Expresión Lógica	Resultado
True	True	P or Q	True
True	False	P or Q	True
False	True	P or Q	True
False	False	P or Q	False

Tabla de Verdad - not

P	Expresión Lógica	Resultado
True	not P	False
False	not P	True

Uso Operadores Booleanos

Si las siguientes variables almacenan :

```
Nota1 = 3.0
Nota2 = 5.5
Nota3 = 7.0
```

¿ Cuáles serían los valores de verdad resultantes al evaluar las siguientes expresiones lógicas ?

- `(Nota1 >= 4.0) and (Nota2 >= 4.0) and (Nota3 >= 4.0)`
- `(Nota1 != Nota2) or (Nota1 < Nota3)`
- `not(Nota1 == Nota2)`

Resumen de Operadores en Python

Operación	Operador	Aridad	Asociatividad	Precedencia
Exponenciación	`**`	Binario	Por la derecha	1
Identidad	`+`	Unario	—	2
Cambio de signo	`-`	Unario	—	2
Multiplicación	`*`	Binario	Por la izquierda	3
División	`/`	Binario	Por la izquierda	3
División entera	`//`	Binario	Por la izquierda	3
Módulo (o resto)	`%`	Binario	Por la izquierda	3
Suma	`+`	Binario	Por la izquierda	4
Resta	`-`	Binario	Por la izquierda	4
Igual que	`==`	Binario	—	5
Distinto de	`!=`	Binario	—	5
Menor que	`<`	Binario	—	5
Menor o igual que	`<=`	Binario	—	5
Mayor que	`>`	Binario	—	5
Mayor o Igual que	`>=`	Binario	—	5
Negación	`not`	Unario	—	6
Conjunción	`and`	Binario	Por la izquierda	7
Disyunción	`or`	Binario	Por la izquierda	8

Tabla 2.4: Características de los operadores Python. El nivel de precedencia 1 es el de mayor prioridad.

FUNCIONES INTEGRADAS

Funciones Integradas (Built-In)

- El intérprete de **Python** tiene una serie de **funciones** y **tipos** incluidos que están **siempre disponibles.**

- Una **función** es un conjunto de **líneas de código** que realizan una **tarea específica** y puede **retornar un valor.**

Algunas **funciones integrada de Python** son :

  - Función de escritura : **print()**
  - Función de lectura : **input()**
  - Funciones : **int()**, **float()**, **str()**

FUNCIÓN DE ESCRITURA

Función de Escritura

- Para poder escribir los **resultados** de nuestros programas por pantalla **Python** tiene una **función** llamada **print.**

- **Su Sintaxis de llamada es** :

print(argumento1, … ,argumentoN)

- Por defecto cada vez que se ejecuta **print** lo que escribe en pantalla parte en **una línea nueva.** Ya que **print** agrega al final un **carácter especial** llamado **terminador/fin de línea.** Esto **podemos evitarlo** si indicamos que no hay terminador de línea , agregando un argumento especial: **end = ‘’**

Ejemplos de Uso función print()

- Para imprimir la cadena de caracteres **HOLA MUNDO** en la pantalla tendríamos que ejecutar la siguiente instrucción :

```
print(“HOLA MUNDO”)
```

- La siguiente secuencia de instrucciones permite almacenar un valor de tipo cadena en una variable llamada **nombre** y luego imprimir un saludo por pantalla a ese nombre.

```
nombre = “JUAN”
print(“HOLA”, nombre)
```

FUNCIÓN DE LECTURA

Función de Lectura (1/2)

- Para poder leer los **datos de entrada** desde teclado que necesiten procesar nuestros programas **Python** tiene una **función** llamada **input.**

- **Su Sintaxis de llamada es :**

```
variable = input([mensaje])
```

- Si el argumento **mensaje** está presente, éste se escribe en la salida estándar sin una nueva línea a continuación.

- **En modo interactivo :** esta **función detiene la ejecución** del **programa** y espera a que **el usuario ingrese un dato y pulse la tecla ENTER**, en ese momento prosigue la **ejecución** del **programa** y la **función** devuelve una **cadena de caracteres.**

Función de Lectura (2/2)

- Por defecto la función **input** retorna el valor leído desde el **teclado** como una **cadena de caracteres.**
- Si deseamos almacenar esa **cadena de caracteres** en una **variable** de tipo **entero** debemos transformarla con la función **int.**
- Si deseamos almacenar esa **cadena caracteres** en una variable de tipo **real** debemos transformarla con la función **float.**

Función int

Sintaxis de llamada : **int**(valor)

- Esta **función** recibe como **argumento** un **valor** y lo retorna convertido a **entero.**

Ejemplos

```
x = 123.567
y = int(x)

a = "123"
b = int(a)
```

Función float

Sintaxis de llamada : **float**(valor)

- Esta **función** recibe como **argumento** un **valor** y lo retorna convertido a **punto flotante.**

Ejemplos

```
x = 123
y = float(x)

a = "123.567"
b = float(a)
```

Ejemplo Uso Función de Lectura

```
nombre = input('Ingrese su nombre : ')
edad = int(input('Ingrese su edad : '))
peso = float(input('Ingrese su peso en Kgs : '))

print('Te llamas', nombre)
print('Tienes',edad,'años')
print('Pesas',peso,'kilogramos')
```

COMENTARIOS

Comentarios

- Los **comentarios**, sirven para **documentar** nuestros **programas fuentes** y son **ignorados** por el **traductor.**

Comentarios

En **Python** existen **2 formas de escribir comentarios :**

- Escribiendo el símbolo **gato** (#) delante de la línea de texto donde está el **comentario.**

```
# Este es un comentario de una línea
```

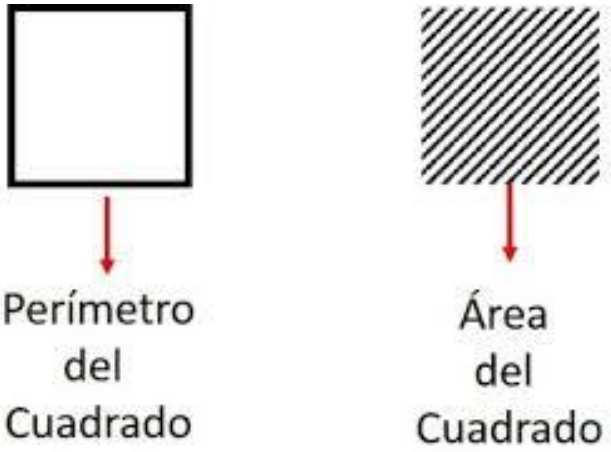
- Escribiendo **triple comilla doble** (“””) al **principio** y al **final** del **comentario.**

```
"""  Este es un comentario
     de tres
     líneas """
```

PRIMER PROGRAMA EN PYTHON

Problema : Área y Perímetro Cuadrado

- Escriba un programa que lea la **medida** en centímetros del **lado** de un **cuadrado**, calcule y muestre por pantalla su **área** y su **perímetro**.



Análisis del Problema

Datos de Entrada

- Medida del lado del cuadrado en centímetros

Datos de Salida

- Área del cuadrado
- Perímetro del cuadrado

Método de Solución

- **Leer** medida del lado en centímetros
- **Calcular** Área y Perímetro
- **Escribir** Área y Perímetro

Diseño del Algoritmo

1. **Leer** medida del lado en centímetros del cuadrado desde **teclado** y almacenarla en la variable **medida**

2. **Calcular** área del cuadrado con fórmula y almacenar resultado en variable **área**

3. **Calcular** perímetro del cuadrado con fórmula y almacenar resultado en variable **perímetro**

4. **Escribir** en pantalla el valor de la variable **área**

5. **Escribir** en pantalla el valor de la variable **perímetro**

Algoritmo en Pseudolenguaje

```
Algoritmo cuadrado
    Leer medida
    area ← medida * medida
    perimetro ← 4 * medida
    Escribir area
    Escribir perimetro
Fin Algoritmo
```

Programa en Python 3 - interactivo

```
1  # Entrada
2  medida = float(input("ingrese medidad del lado del cuadrado: "))
3
4  # Proceso
5  area = medida * medida
6  perimetro = 4 * medida
7
8  # Salida
9  print("Área Cuadrado =",area)
10 print("Perímetro Cuadrado =",perimetro)
```

Programa en Python 3 - interactivo

```
1   # Entrada
2   medida = float(input("ingrese medidad del lado del cuadrado: "))
3
4   # Proceso
5   area = medida * medida
6   perimetro = 4 * medida
7
8   # Salida
9   print("Área Cuadrado =",area)
10  print("Perímetro Cuadrado =",perimetro)
```

```
ingrese medidad del lado del cuadrado: |
```

```
ingrese medidad del lado del cuadrado: 10
Área Cuadrado = 100.0
Perímetro Cuadrado = 40.0
```

Programa en Python 3 – NO intercativo

```
1  # Entrada
2  medida = float(input())
3
4  # Proceso
5  area = medida * medida
6  perimetro = 4 * medida
7
8  # Salida
9  print("Área Cuadrado =",area)
10 print("Perímetro Cuadrado =",perimetro)
```

Programa en Python 3 – NO interactivo

```python
# Entrada
medida = float(input())

# Proceso
area = medida * medida
perimetro = 4 * medida

# Salida
print("Área Cuadrado =",area)
print("Perímetro Cuadrado =",perimetro)
```

Stdin Inputs

```
10
```

```
Área Cuadrado = 100.0
Perímetro Cuadrado = 40.0
```