El futuro digital es de todos

MinTIC

Validaciones, Excepciones

Funciones

Estructuras de Datos

Misión TIC 2022

unab
VIGILADA MINEDUCACIÓN

Misión TIC 2022

Temas – Sesión 1

- Validación - Excepciones
- Funciones
- Estructura de Datos
  - Conceptualización
  - Listas

Estructuras de control iterativas

Validaciones y Excepciones

Validaciones y excepciones

Validación de información de entrada

En la entrada o ingreso de la información, se debe aplicar un proceso de validación, que consiste en verificar su cumpla con el tipo de dato, especialmente para los numéricos y si deben cumplir con un rango de valores o característica especial.

Validaciones y excepciones

Validación de información de entrada – Tipo de Datos int-float

Variable de control - Bandera

```
while True:
        try:
            x = int(input("Ingrese un numero: "))
            break
        except ValueError:
            print("Oops!  No es un Entero.  Intenta de nuevo...")
```

Si se presenta ERROR y n0 permite salir del ciclo

Salir del ciclo WHILE

Validaciones y excepciones

Validación de información de entrada – Tipo de Datos int-float y rangos

Variable de control - Bandera

```
# Validación Categoria(Entero y valor1,2 o 3)
while True:
    try:
        categoria=int(input("Categoría(1,2,3): "))
        if categoria<1 or categoria>3:
            print("categoria debe ser1,2 o 3")
            continue

        break
    except ValueError:
        print("Categoria debe ser un dato entero")

print("Proceso Finalizado")
```

Regrese al ciclo WHILE

Salir del ciclo WHILE

Si se presenta ERROR y n0 permite salir del ciclo

Ejercicios

Ejercicio

Dado el nombre y estrato (1,2,3,4,5) de un usuario del servicio de energía eléctrica, calcular lo que pagaría de tarifa básica del servicio de energía eléctrica, que depende del estrato, así

Estrato	Tarifa Básica
1	$10.000
2	$15.000
3	$30.000
4	$50.000
5	$65.000

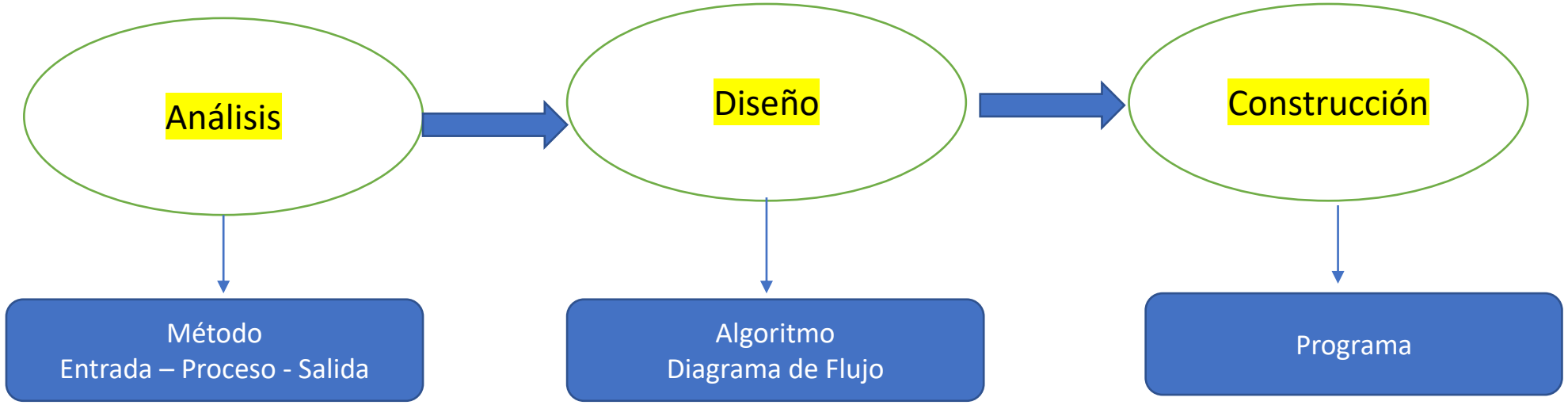
Se pide visualizar el nombre y tarifa básica

Validación entrada de información

Ejercicios

Ejercicio

Metodología -> Pensamiento lógico estructurado



Validación entrada de información

Ejercicios

Ejercicio

Análisis –> Método Entrada-Proceso-Salida

Operaciones, cálculos, estructuras de control

Entrada
Se debe LEER

Proceso

Salida Se debe
IMPRIMIR

nombre, estrato

1

Condicional:
Si el estrato es 1,2,3,4,5 toma un valor de traifa_basica

2

3

nombre, tarifa_basica

Validación entrada de información

Ejercicios

Ejercicio

```
Algoritmo Calcular_tarifa_basica
    Escribir "Nombre usuario: "
    Leer nombre
    Escribir "Estrato(1,2,3,4 o 5): "
    Leer estrato
    Si estrato=1 Entonces
        tarifa_basica=10000
    SiNo
        Si estrato=2 Entonces
            tarifa_basica=15000
        SiNo
            Si estrato=3 Entonces
                tarifa_basica=30000
            SiNo
                Si estrato=4 Entonces
                    tarifa_basica=50000
                SiNo
                    tarifa_basica=65000
                Fin Si
            Fin Si
        Fin Si
    Fin Si
    Escribir "Nombre usuario: ",nombre
    Escribir "Tarifa Básica: ",tarifa_basica
FinAlgoritmo
```

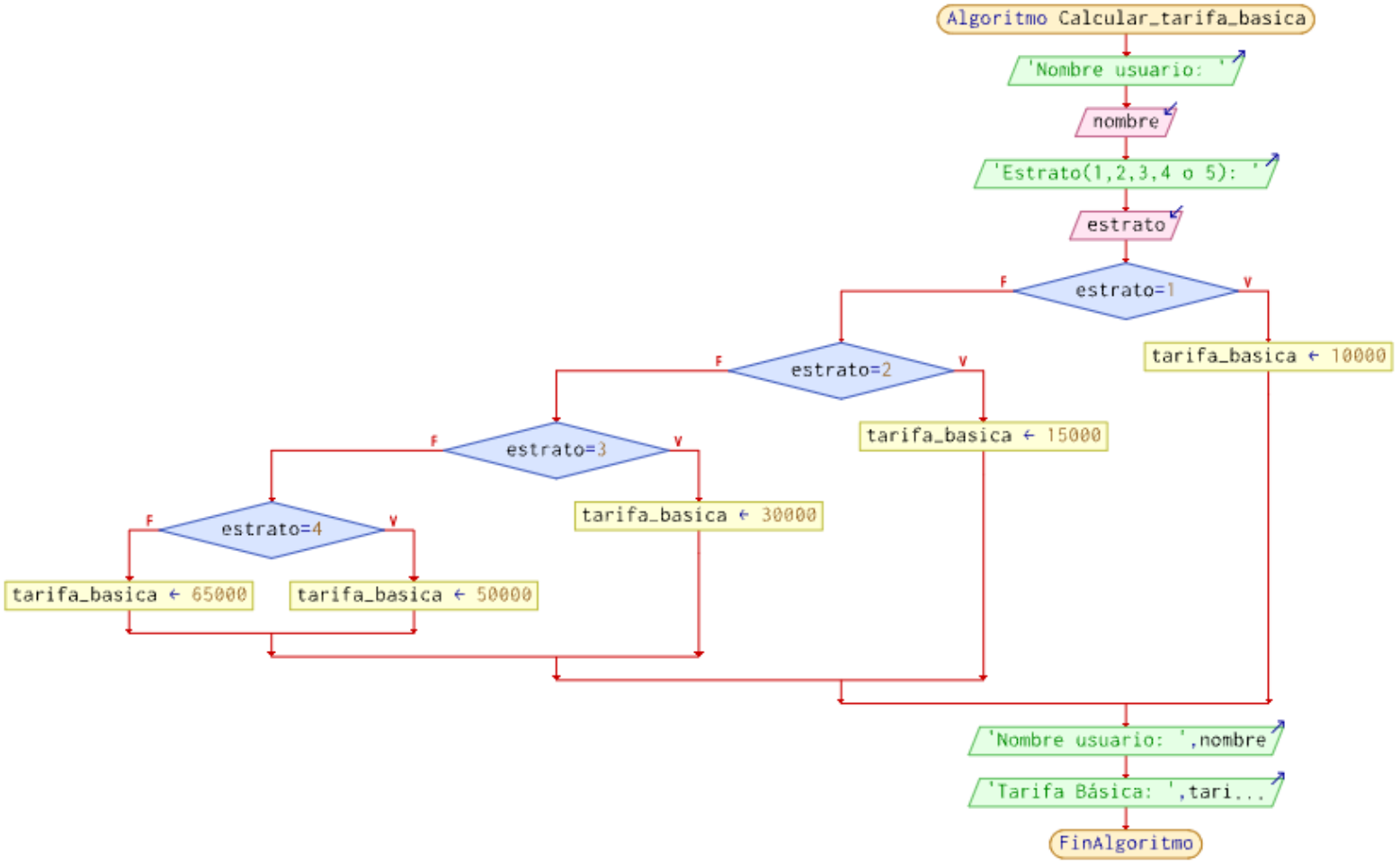
```
PSeInt - Ejecutando pr...
*** Ejecución Iniciada. ***
Nombre usuario:
> Sergio Medina
Estrato(1,2,3,4 o 5):
> 4
Nombre usuario: Sergio Medina
Tarifa Básica: 50000
*** Ejecución Finalizada. ***
No cerrar esta ventana   Siempre visible   inic
```

Diseño –> Algoritmo

Validación entrada de información

Ejercicios

Ejercicio



Diseño –> Diagrama de flujo

Validación entrada de información

Ejercicios

Ejercicio

```
#Programa para calcular la tarifa básica
#Autor: Sergio Medina
#Fecha: 29/04/2022

nombre=input("Nombre usuario: ")
#Validación estrato (tipo de dato y rango)
while True:
    try:
        estrato=int(input("Estrato(1,2,3,4 o 5): "))
        if estrato<1 or estrato>5:
            print("Estrato debe ser 1,2,3,4 o 5")
            continue
        break
    except ValueError:
        print("El estrato debe ser un dato entero")
if estrato==1:
    tb=10000
elif estrato==2:
    tb=15000
elif estrato==3:
    tb=30000
elif estrato==4:
    tb=50000
else:
    tb=65000
print("Nombre: ",nombre)
print("Tarifa Básica: ",'{:,.2f}'.format(tb))
```

Construcción –> Programa

Funciones

Funciones

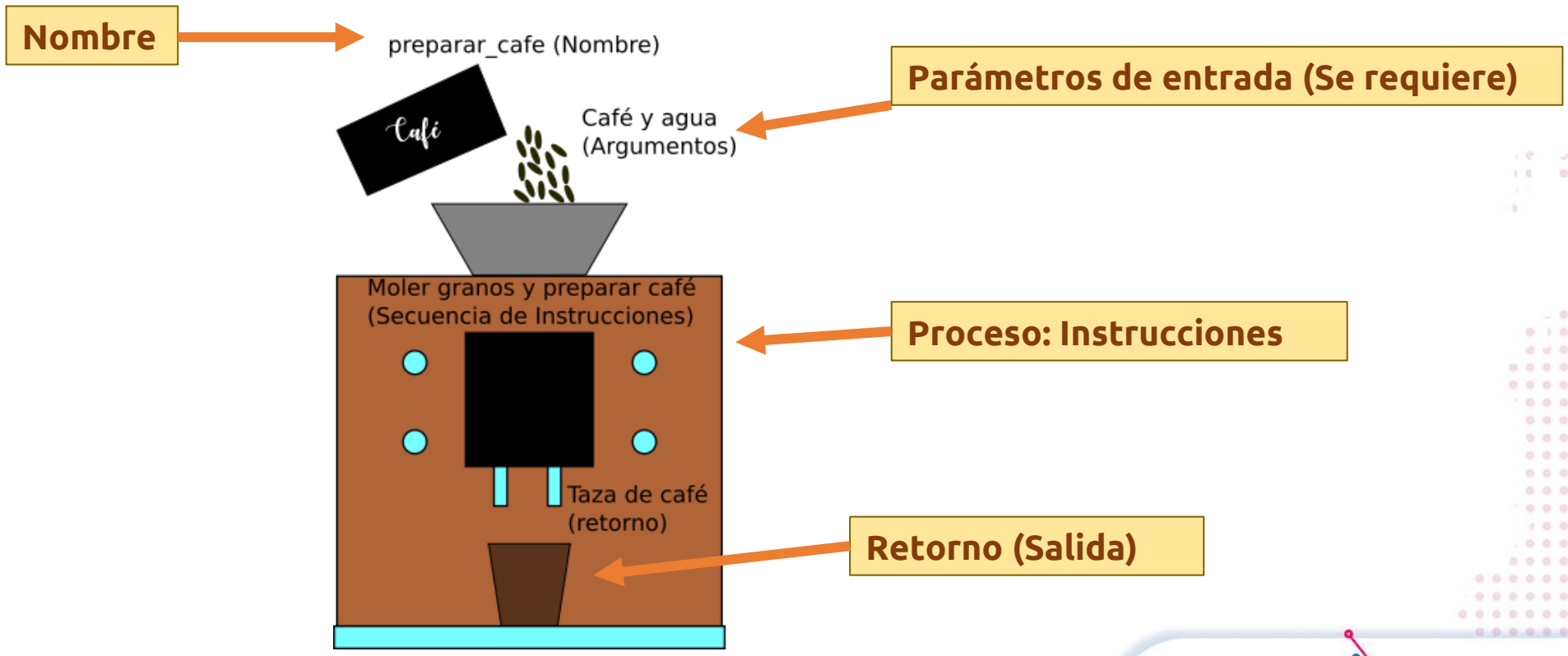
Conceptualización

¿Has visto alguna vez una carrera de autos de fórmula uno? Pues bien, hay un momento en la competición en la que los autos deben entrar a pits. La razón es que al auto se le debe hacer un mantenimiento a las llantas y se le debe suministrar combustibles. Ambas funciones deben llevarse a cabo luego de un determinado número de vueltas, cuando el ingeniero automovilístico encargado lo determine. Supongamos que la función general de la entrada a pits es realizar ambas tareas (cambio de llantas y suministrar combustible) ejecutadas una seguida de la otra. Así pues, una función se puede definir como una secuencia de instrucciones que tiene como finalidad llevar a cabo una tarea específica; como por ejemplo, realizar la suma de dos números, contar las palabras de una cadena de caracteres, etc.

Las **funciones en programación reciben un nombre que debe ser coherente con su función**. Por lo general, un programa es dividido por diferentes tipos de funciones, que llevan a cabo diferentes tipos de tareas, de tal manera que se logra la solución de un problema más grande. Adicionalmente, así como un auto de carreras entre a la zona de pits las veces que sea necesario, en un programa, las funciones son usadas las veces que se desee, esto es son estructuras de códigos reutilizables!

Funciones

Estructura de una función



Estructura básica de una función haciendo analogía a la preparación de una taza de café.

Funciones y módulos predeterminados

En los lenguajes de programación, se puede definir cualquier función que se desee; pero, para facilitarnos un poco la vida existen funciones predefinidas, o sea que alguien más ya las creó y fueron incorporadas en el lenguaje de programación, en este caso **Python**.

En **Python**, se pueden encontrar dos tipos de funciones predeterminadas, las cuales son:

- **Funciones predefinidas**.
- **Los módulos predefinidos**. son archivos que contienen **métodos predefinidos** (funciones), que no pueden existir por sí solos, ya que se encuentran asociados a determinado **objeto** o tipo de dato (listas, cadenas, caracteres, etc), de tal manera que, operan sobre ellos.

Funciones

Funciones y módulos predeterminados

```
>>> max(40,30,5,90,20)
90
```

```
>>> min(40,30,5,90,20)
5
```

Funciones

```
>>> numeros=[11,12,31,41,52,63,78]
>>> len(numeros)
7
```

Librerías: Agrupan , funciones, métodos, etc.

```
>>> import math
>>> math.sqrt(16)
4.0
```

Método: Operación

```
>>> numeros=[6,3,1,4,5,2]
>>> numeros_ordenados=sorted(numeros)
>>> numeros_ordenados
[1, 2, 3, 4, 5, 6]
```

```
>>> numeros=[6,3,1,4,5,2]
>>> numeros_ordenados=sorted(numeros,reverse=True)
>>> numeros_ordenados
[6, 5, 4, 3, 2, 1]
```

Funciones y módulos predeterminados

```
# Programa para hallar la raíz cuadrada de un número
# Autor: Sergio Medina
# Fecha: 09/05/2022

import math
x=int(input("Valor: "))
print("Raiz cuadrada de ",x," es: ",math.sqrt(x))
```

Funciones y métodos – Cadenas de caracteres

```
>>> numeros=[1,2,3,4,5,6,1,7,1]
>>> print(numeros.count(1))
3
```

```
>>> letras=["a","e","t","v","u","Z","c","a"]
>>> print(letras.count("a"))
2
```

Contar en listas

```
>>> items="1,2,3,4,5,6"
>>> items.split(",")
['1', '2', '3', '4', '5', '6']
>>> items.split(",")[4]
'5'
```

```
>>> items="la casa de Luisa es muy bonita"
>>> items.split()
['la', 'casa', 'de', 'Luisa', 'es', 'muy', 'bonita']
```

Crea Lista desde una cadena , su argumento es el separador de elementos de la lista creada

Cantidad de palabras en una frase

```
>>> items="la casa de Luisa es muy bonita"
>>> items.split()
['la', 'casa', 'de', 'Luisa', 'es', 'muy', 'bonita']
>>> len(items.split())
7
```

Funciones y métodos – Cadenas de caracteres

```
# Programa para calcular la cantidad de palabras de un frase
# Autor: Sergio Medina
# Fecha: 10/05/2022

frase=input("Frase: ")
can_palabras=len(frase.split())
print("Cantidad de palabras: ",can_palabras)
```

Modularidad: Acoplamiento y Cohesión de módulos

Modularidad: Acoplamiento y Cohesión de módulos

Modularidad

Uno de los aspectos fundamentales de la programación moderna, base de los nuevos paradigmas, es sin duda alguna la **modularidad**, entendida como la generación de módulos o segmentos funcionales e independientes que permitan una mejor organización y compresión de un programa. Este aspecto se basa en la aplicación de dos técnicas propias de la ingeniería del software, denominadas Acoplamiento de módulos y Cohesión de módulos que definen unas guías en la definición de un módulo.

Modularidad: Acoplamiento y Cohesión de módulos

Cohesión de módulos

La técnica de la ingeniería del software, denominada Cohesión de Módulos busca medir el grado de relación ò dependencia que existe entra las actividades propias de un proceso o módulo. La finalidad es generar módulos que realicen un proceso determinado y por consiguiente las actividades o instrucciones que contine están todas relacionadas con el objetivo del módulo. Por ejemplo, un módulo de liquidación de comisiones, solo debe contener las instrucciones que permitan el calculo del valor de la comisión y no incluir otro tipo instrucciones, como las de incrementar contadores y sumadores.

Modularidad: Acoplamiento y Cohesión de módulos

Acoplamiento de módulos

La técnica La técnica del Acoplamiento de Módulos que se aplica después de la cohesión, tiene como objetivo la generación de módulos independientes dentro de un proceso, en los cuales, cada uno de ellos define sus propias variables y la comunicación con ellos se realice a través de parámetros, o sea, variables (argumentos) que recibe el módulo que le permitan realizar la función especifica para lo que fue definido. Los módulos independientes, que reciben parámetros de entrada y retornan una salida específica, permiten su reutilización en otros programas y procesos, lo que facilita el desarrollo de software.

Funciones

Funciones y Argumentos (Análisis – Diseño – Construcción)

Modularidad: Acoplamiento y Cohesión de módulos

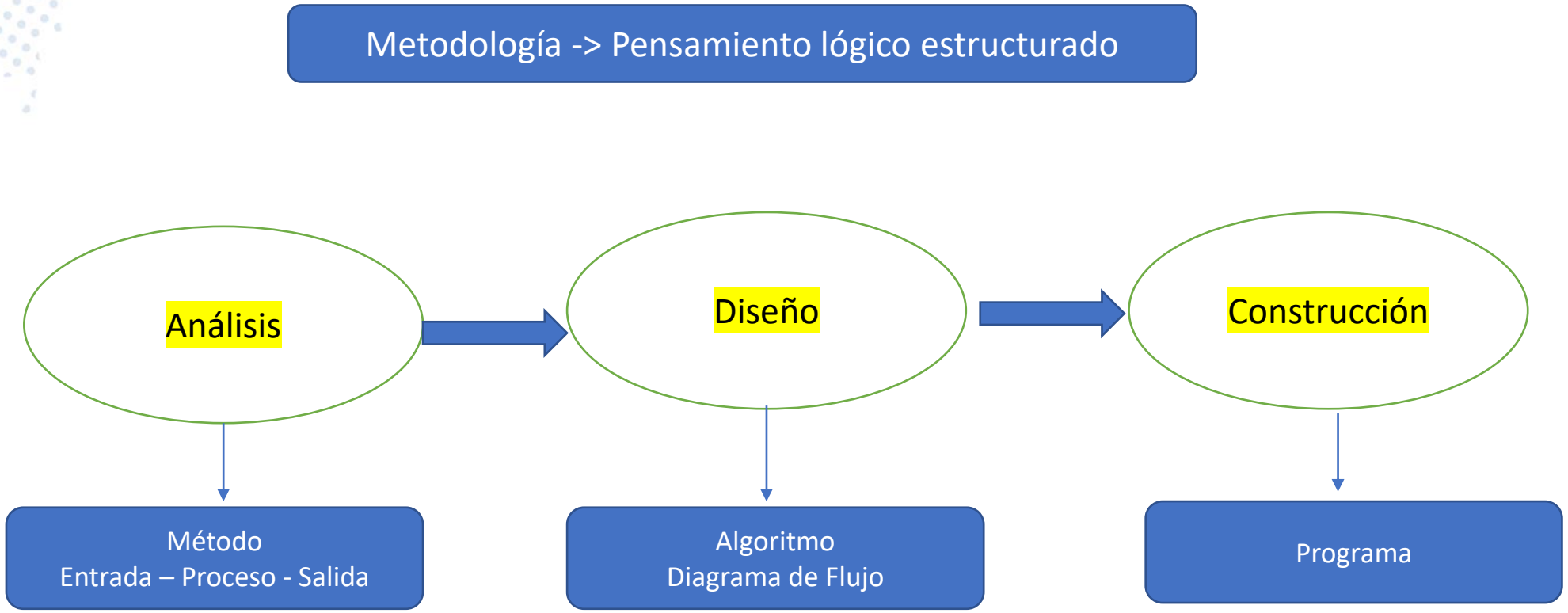
La empresa de teléfonos de la ciudad necesita realizar su proceso de facturación en forma automática, contando con los N abonados, de los cuales conoce el nombre, estrato, que puede ser (1, 2, 3, 4, 5), cantidad de impulsos del mes (N es suministrado). Además la empresa nos informa que para la liquidación de la factura se debe tener en cuenta el valor de la tarifa básica, de acuerdo al estrato, que depende de la siguiente tabla:

Estrato	Ttarifa Básica
1	$10.000
2	$15.000
3	$20.000
4	$25.000
5	$30.000

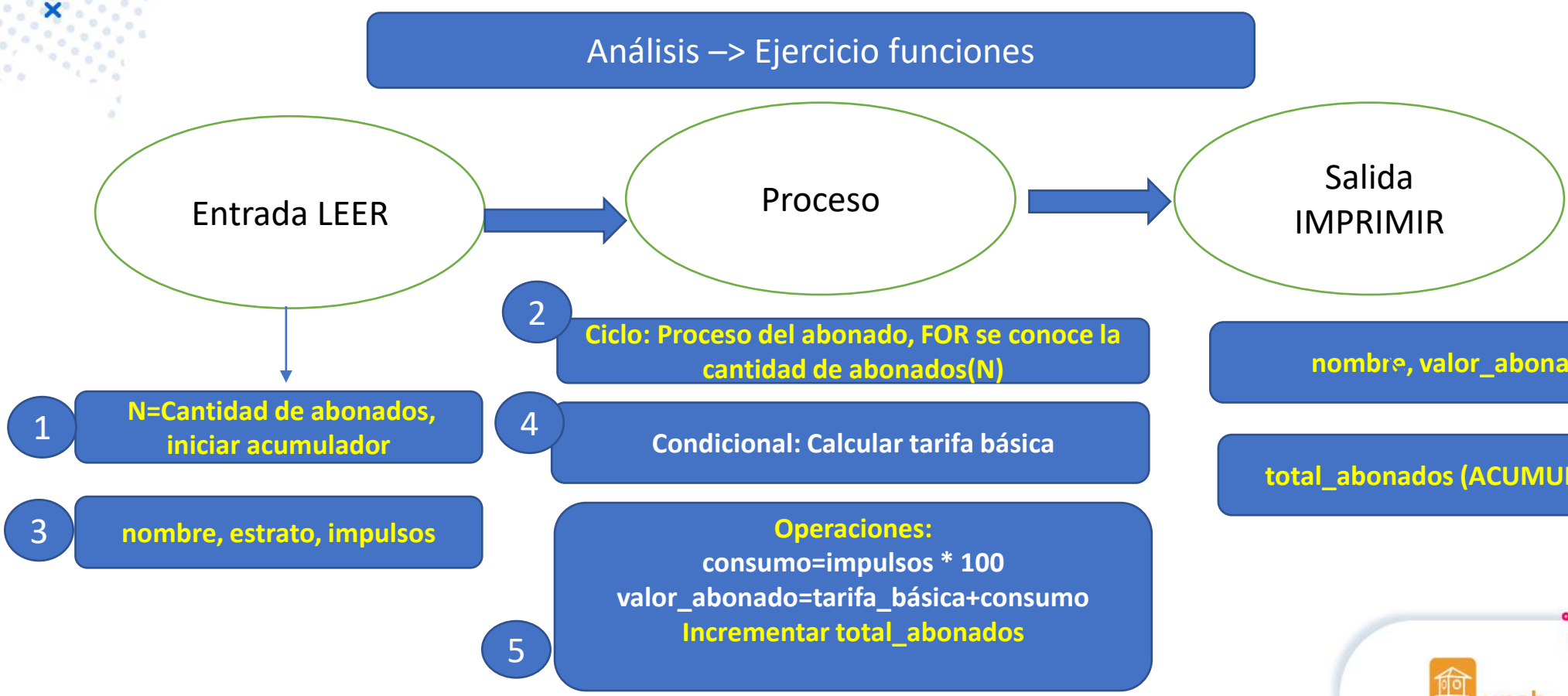
Además se debe calcular el valor de los impulsos, con base en la cantidad de impulsos del mes, conociendo que cada impulso tiene un valor de $100. Con esta información, se desea:

- ☑ Valor a pagar de cada abonado.
- ☑ Valor total a pagar(Todos los abonados)

Modularidad: Acoplamiento y Cohesión de módulos



Modularidad: Acoplamiento y Cohesión de módulos



Modularidad: Acoplamiento y Cohesión de módulos

Análisis - Modularidad

Parámetros de entrada

estrato, impulsos

MODULO (FUNCION)
proceso_abonado

Calcular tarifa básica
Calcular el consumo
Calcular valor_abonado (operación de suma)

Parámetros de salida

valor_abonado

FUNCIÓN retorna o regresa un solo valor

Parte de la 4 5

Modularidad: Acoplamiento y Cohesión de módulos

```
Funcion valor_abonado <- proceso_abonado ( estrato,impulsos )
    Si estrato=1 Entonces
        tarifa_basica=10000
    SiNo
        Si estrato=2 Entonces
            tarifa_basica=15000
        SiNo
            Si estrato=3 Entonces
                tarifa_basica=20000
            SiNo
                Si estrato=4 Entonces
                    tarifa_basica=25000
                SiNo
                    tarifa_basica=30000
                Fin Si
            Fin Si
        Fin Si
    Fin Si
    consumo=impulsos*100
    valor_abonado=tarifa_basica+consumo
Fin Funcion
```

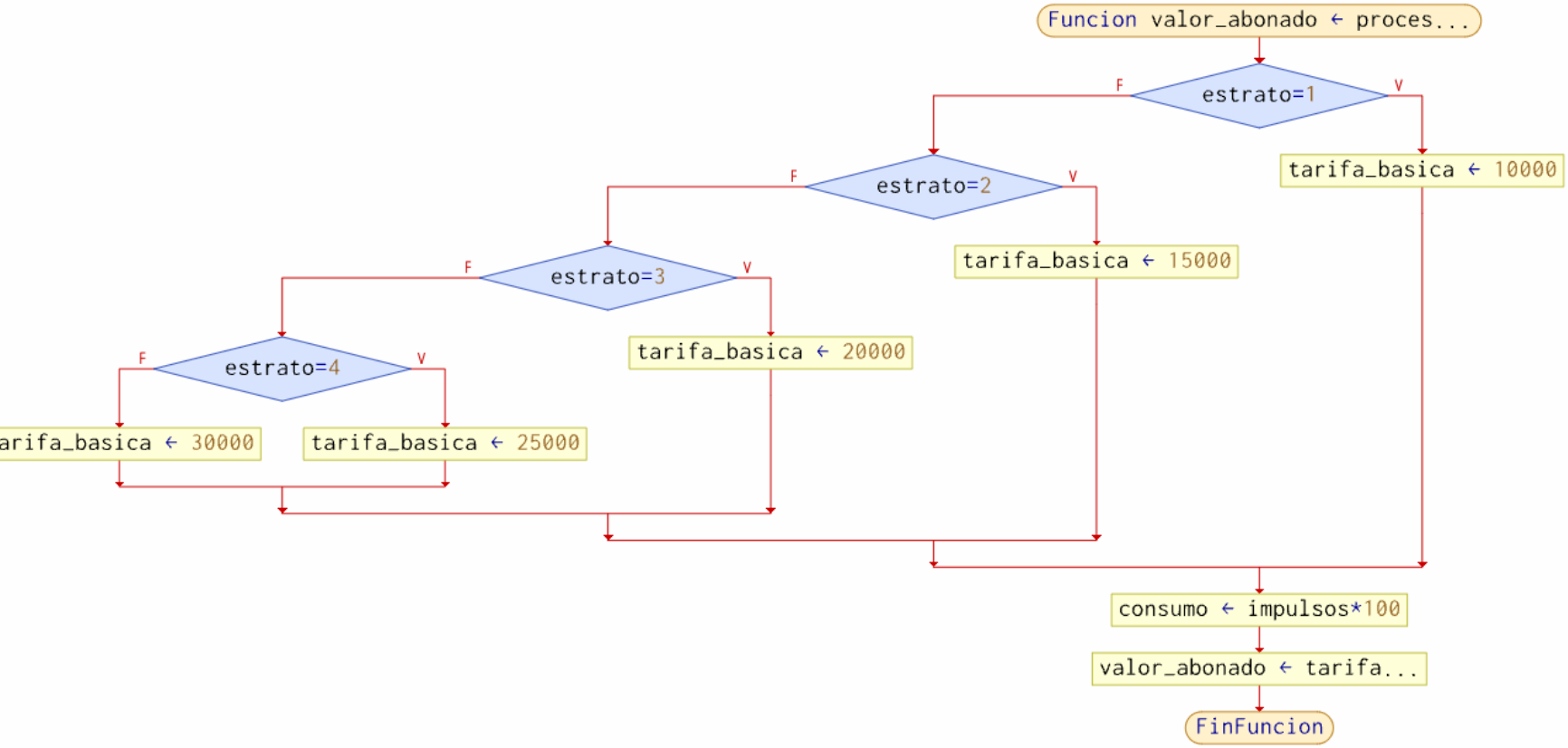
Diseño – Algoritmo Función

Modularidad: Acoplamiento y Cohesión de módulos

```
Algoritmo servicio_telefono
    Escribir "Cantidad de abonados: "
    Leer N
    total_abonados=0
    Para i<-1 Hasta N Con Paso 1 Hacer
        Escribir "Nombre: "
        Leer nombre
        Escribir "Estrato (1,2,3,4,5): "
        Leer estrato
        Escribir "Impulsos del mes: "
        Leer impulsos
        valor_abonado=proceso_abonado(estrato,impulsos)
        total_abonados=total_abonados+valor_abonado
        Escribir "Nombre abonado: ",nombre
        Escribir "Valor servicio abonado: ",valor_abonado
    Fin Para
    Escribir "Valot total abonados: ",total_abonados
FinAlgoritmo
```

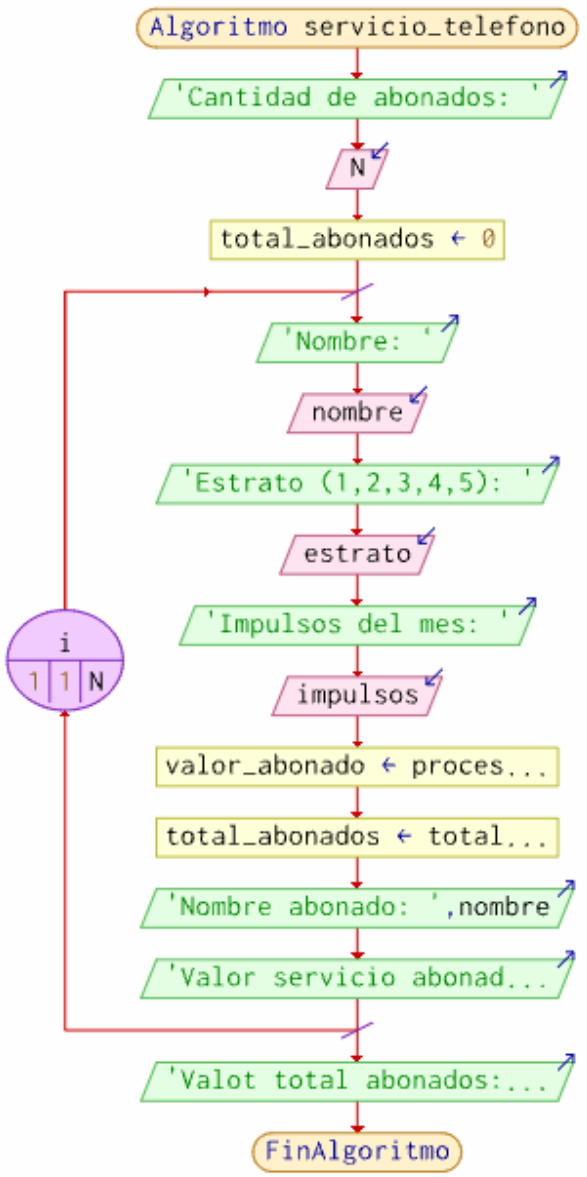
Diseño – Algoritmo Principal

Modularidad: Acoplamiento y Cohesión de módulos



Diseño – Diagrama de flujo (función)

Modularidad: Acoplamiento y Cohesión de módulos



Diseño – Diagrama de flujo (Principal)

Modularidad: Acoplamiento y Cohesión de módulos

```
# Programa liquidación servicio abonados con funciones
# Autor: Sergio Medina
# Fecha: 11/05/2022

# Funciones
def proceso_abonado(estrato,impulsos):
    if estrato==1:
        tarifa_basica=10000
    elif estrato==2:
        tarifa_basica=15000
    elif estrato==3:
        tarifa_basica=20000
    elif estrato==4:
        tarifa_basica=25000
    else:
        tarifa_basica=30000
    consumo=impulsos*100
    valor_abonado=tarifa_basica+consumo
    return valor_abonado
```

Construcción –> Programa Funciones, versión 1

Modularidad: Acoplamiento y Cohesión de módulos

```
# Programa principal
N=int(input("Cantidad de abonados: "))
total_abonados=0
for i in range(N):
    nombre=input("Nombre abonado: ")
    estrato=int(input("Estrato (1,2,3,4,5): "))
    impulsos=int(input("Impulsos del mes: "))
    #Llamado a la función
    valor_abonado=proceso_abonado(estrato,impulsos)
    total_abonados+=valor_abonado
    print("Nombre abonado: ",nombre)
    print("Valor servicio abonado: ","{:,.2f}".format(valor_abonado))
print("Valor total abonados: ","{:,.2f}".format(total_abonados))
```

Construcción –> Programa principal Versión 1

Modularidad: Acoplamiento y Cohesión de módulos

```
# Programa liquidación servicio abonados con funciones
# Autor: Sergio Medina
# Fecha: 11/05/2022


# Funciones
def valida_entero(etiqueta):
    while True:
        try:
            dato=int(input(etiqueta))
            break
        except ValueError:
            print(etiqueta," debe ser un número entero")
    return dato

def valida_estrato():
    while True:
        try:
            estrato=int(input("Estrato (1,2,3,4,5): "))
            if estrato<1 or estrato>5:
                print("El estrato debe ser 1,2,3,4 o 5")
                continue
            break
        except ValueError:
            print("El estrato debe ser entero")
    return estrato
```

Construcción –> Programa con validación entrada Versión 2 – Funciones de validación

Modularidad: Acoplamiento y Cohesión de módulos

```
def proceso_abonado(estrato,impulsos):
    if estrato==1:
        tarifa_basica=10000
    elif estrato==2:
        tarifa_basica=15000
    elif estrato==3:
        tarifa_basica=20000
    elif estrato==4:
        tarifa_basica=25000
    else:
        tarifa_basica=30000
    consumo=impulsos*100
    valor_abonado=tarifa_basica+consumo
    return valor_abonado
```

Construcción –> Programa con validación entrada Versión 2 – Funciones

Modularidad: Acoplamiento y Cohesión de módulos

```
# Programa principal
N=valida_entero("Cantidad de abonados: ")
total_abonados=0
for i in range(N):
    nombre=input("Nombre abonado: ")
    estrato=valida_estrato()
    impulsos=valida_entero("Impulsos del mes: ")
    #Llamado a la función
    valor_abonado=proceso_abonado(estrato,impulsos)
    total_abonados+=valor_abonado
    print("Nombre abonado: ",nombre)
    print("Valor servicio abonado: ","{:,.2f}".format(valor_abonado))
print("Valor total abonados: ","{:,.2f}".format(total_abonados))
```

Construcción –> Programa principal con validación entrada
Versión 2

Estructura de Datos

Conceptualización Estructura de Datos

Estructura de Datos

Conceptualización

En la vida cotidiana nos vemos enfrentados a crear listas, por ejemplo la lista de útiles para el colegio o la universidad, la lista de personas que se invitará a una fiesta. En la preparación de alimentos, se debe realizar una lista con los ingredientes: carne molida, tomate, pan, cebolla, aceite, queso tajado, lechuga y tocineta.

Las **estructuras de datos** son agrupaciones de variables simples que conforman un conjunto de datos más complejo con el cual puedes dar soluciones eficientes a situaciones más cercanas a la vida práctica, como lo son por ejemplo: el manejo de calificaciones de estudiantes de un curso o la gestión de nómina de los empleados de una empresa.

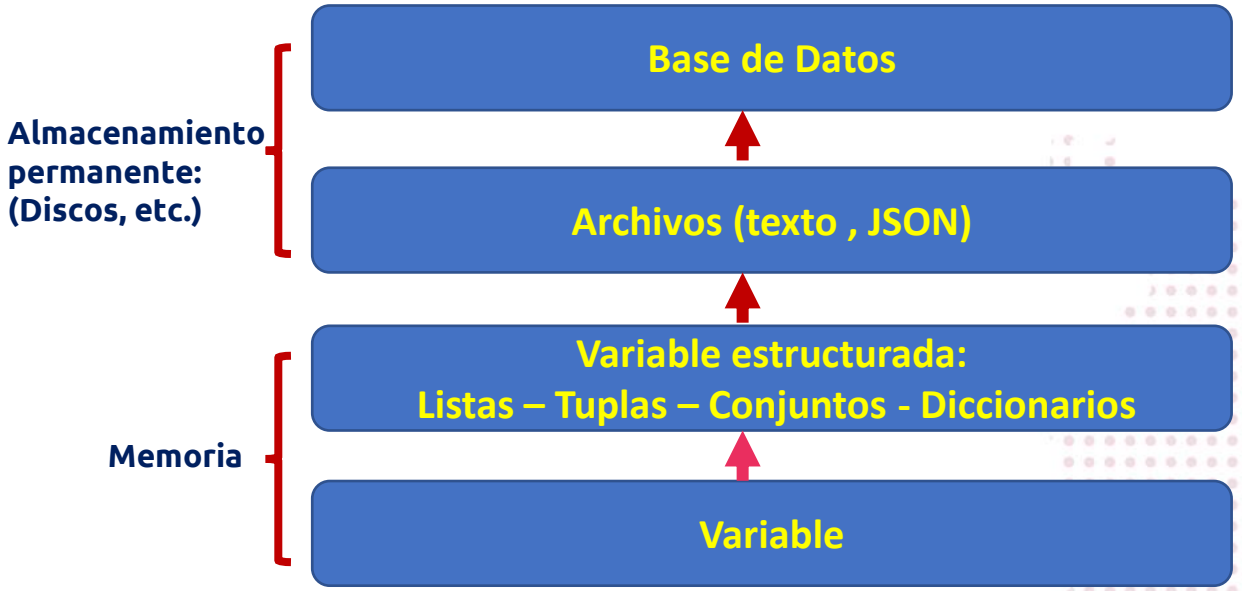
Estructura de Datos

Conceptualización

Variable: Es un espacio de memoria que **contiene un dato simple** de tipo cadena, numérico, booleano, etc.

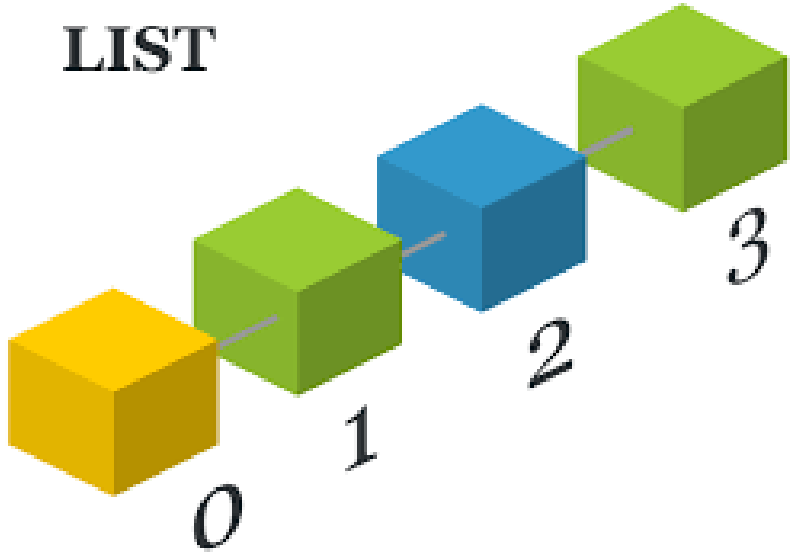
Al contenido de este espacio de memoria, se accede a través de lo que llamamos un **identificador o Nombre de Variable.**

Variable Estructurada: es un agrupamiento, empaquetamiento o colección de varios espacios de memoria, a los cuales se accede a través de un único identificador**.**



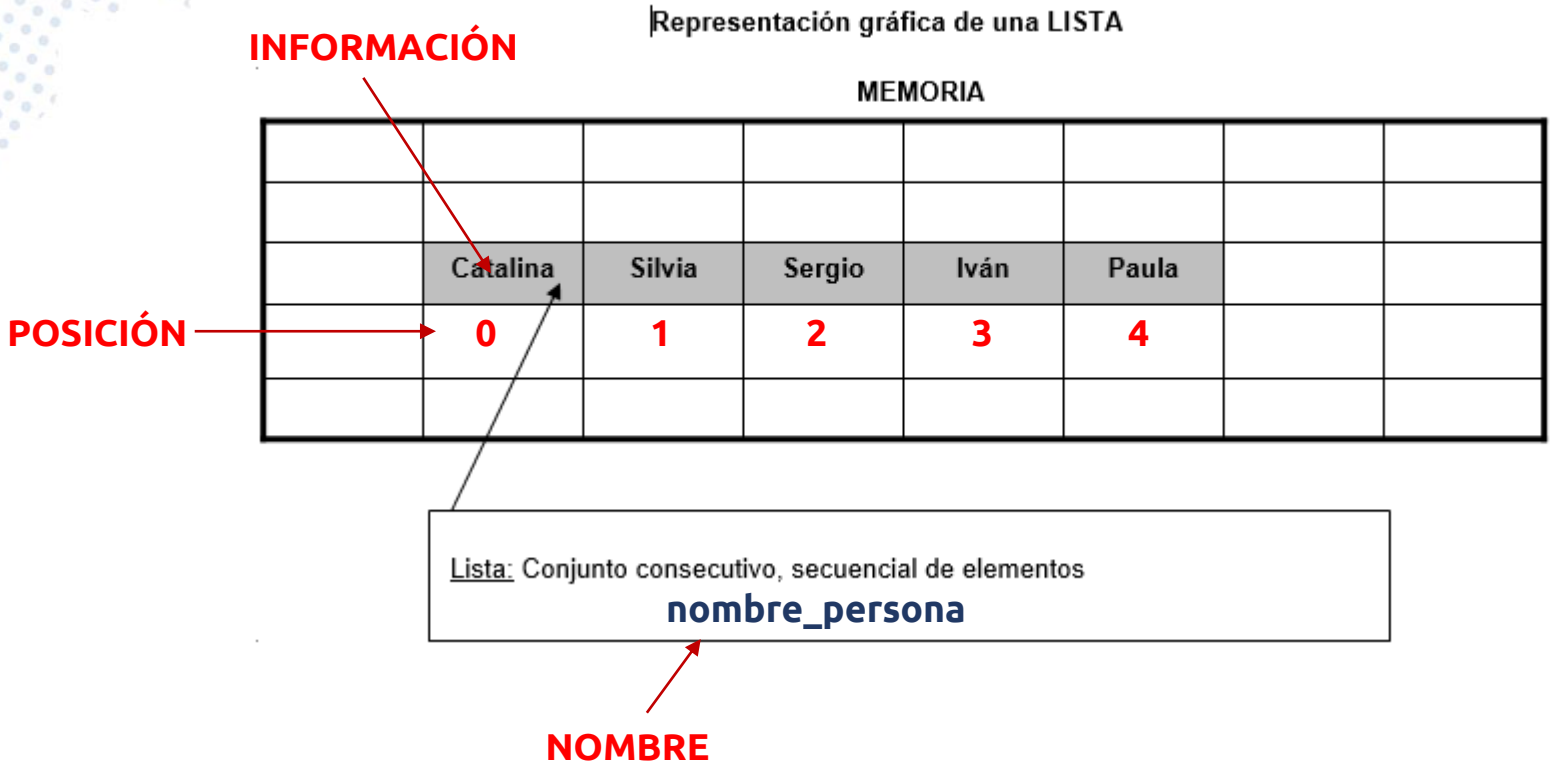
Listas

Las listas son estructuras que permiten ser modificadas a lo largo de la ejecución de un programa usando algunos métodos y operadores. Este comportamiento le da la caracterización a las listas de ser **estructuras mutables**.



Estructura de Datos

Listas - Características



Referenciar elementos

nombre_persona[2] = “Sergio”

nombre_persona[0:2] = “Catalina”,”Silvia”

Estructura de Datos

El futuro digital es de todos | MinTIC

Listas – Creación

```
>>> lista_numeros=[10,15,20,30,40]
>>> lista_numeros[2]
20
>>> lista_nombres=["Sergio","Catalina","Silvia","Iván","Elsa"]
>>> lista_nombres[4]
'Elsa'

>>> lista_nombres[0:2]
['Sergio', 'Catalina']

>>> lista_pares=list(range(2,20,2))
>>> lista_pares
[2, 4, 6, 8, 10, 12, 14, 16, 18]

>>> lista_elementos=[1,"Juan",[2,3],10.4,"Pedro"]
>>> lista_elementos[2]
[2, 3]
>>> lista_elementos[3:5]
[10.4, 'Pedro']
```

Crea la lista con valores desde 2 hasta 20, con incrementos de 2. No se toma el valor final del rango

unab VIGILADA MINEDUCACIÓN

Misión TIC 2022

Listas - Métodos

Podemos crear una lista y luego modificar sus elementos mientras se ejecuta el código.

Para esto, las listas tienen un conjunto de métodos y funciones que realizan acciones y operaciones sobre una lista en particular. Algunos de estos métodos son:

append, extend, insert, pop, remove

El método append permite añadir un ítem al final de una lista

El método extend se utiliza para agregar elementos interables como un string u otra lista separando sus elementos.

El método insert permite añadir un ítem en una posición o índice específico

El método pop quita un elemento de la lista dado su índice.

El método remove para remover un ítem de una lista basado en el valor

Estructura de Datos

El futuro digital es de todos | MinTIC

Listas – Métodos

```
>>> lista=[10,20,"Juan",30,"Sergio"]
>>> lista
[10, 20, 'Juan', 30, 'Sergio']
>>> lista.append(40)
>>> lista
[10, 20, 'Juan', 30, 'Sergio', 40]
>>> lista.append("Paula")
>>> lista
[10, 20, 'Juan', 30, 'Sergio', 40, 'Paula']
>>> lista.extend([60,80])
>>> lista
[10, 20, 'Juan', 30, 'Sergio', 40, 'Paula', 60, 80]
>>> lista.insert(1,"Luis")
>>> lista
[10, 'Luis', 20, 'Juan', 30, 'Sergio', 40, 'Paula', 60, 80]
>>> lista.pop(4)
30
>>> lista
[10, 'Luis', 20, 'Juan', 'Sergio', 40, 'Paula', 60, 80]
>>> lista.remove("Sergio")
>>> lista
[10, 'Luis', 20, 'Juan', 40, 'Paula', 60, 80]
```

unab VIGILADA MINEDUCACIÓN

MisiónTIC2022

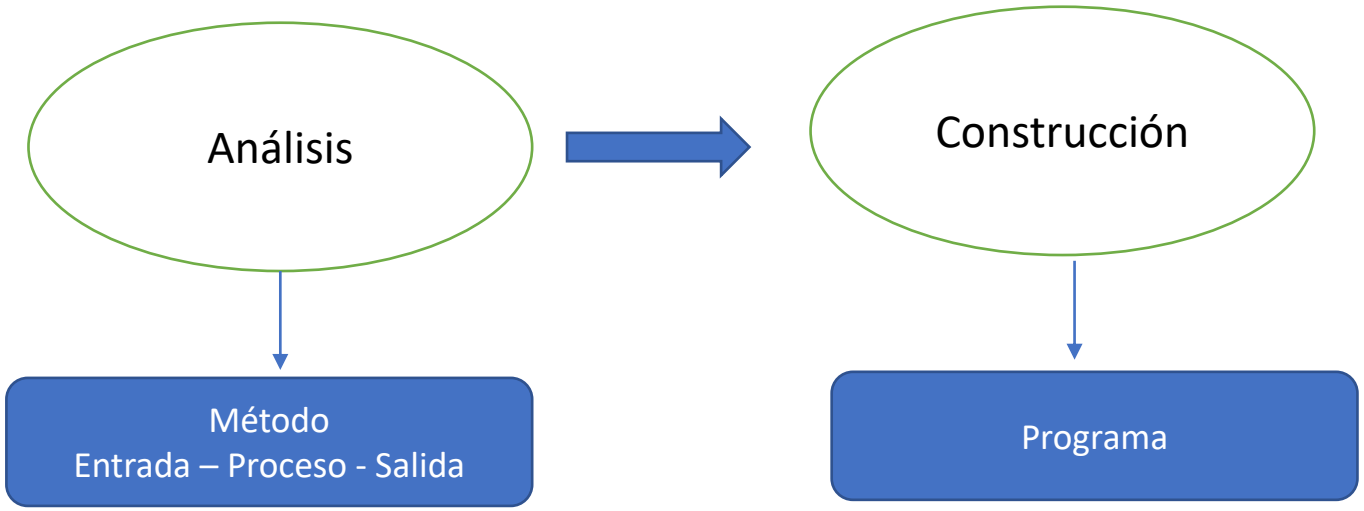
Listas - Ejercicio

Dada una lista con nombres completos de personas, realizar un programa que genere una segunda con la cantidad de palabras de cada uno de los nombres. La lista de nombres debe llenarse a través de nombres que se ingresan por teclado, hasta que el nombre ingresado sea “FIN”

Se debe imprimir la lista de nombres y la lista con la cantidad de palabras de cada nombre.

Listas - Ejercicio

Metodología -> Pensamiento lógico estructurado



Listas - Ejercicio

Listas
Ciclos
- Llenar
- Procesar

Análisis –> Ejercicio Listas

Entrada LEER → Proceso → Salida IMPRIMIR

1. nombre
2. Ciclo: Llenar la lista de nombres (WHILE mientras nombre no sea FIN)
3. Ciclo: Recorrer lista para calcular cantidad de palabras de cada nombre (FOR porque se conoce la cantidad de elementos
4. Contar las palabras de cada elemento de la lista de nombres (len y Split)
5. Lista de nombres y de cantidad de palabras

Listas - Ejercicio

Análisis - Modularidad

Parámetros de entrada

nombre

MODULO (FUNCION)
contar_palabras:

Uso del split y el len

Parámetros de salida

Cantidad_palabras

FUNCIÓN retorna o regresa un solo valor

Listas - Ejercicio

Construcción - Programa

```
# Programa para generar  a partir de una lista de nombres otra lista
# con la cantidad de palabras
# Autor: Sergio Medina
# Fecha: 13/05/2021

# Funciones
def contar_palabras(nombre):
    cant_palabras=len(nombre.split())
    return cant_palabras
# Crear listas vacias
lista_nombres=[]
lista_palabras=[]
# LLenar lista de nombres (Leer)
nombre=input("Nombre completo: ")
while nombre!="FIN":
    lista_nombres.append(nombre)
    nombre=input("Nombre completo: ")
# Procesar Lista
for x in lista_nombres:
    cant_palabras=contar_palabras(x)
    lista_palabras.append(cant_palabras)
# Imprimir listas
print(lista_nombres)
print(lista_palabras)
```

Estructura de Datos

Ejercicios Propuestos

El futuro digital es de todos

MinTIC

Validaciones, Excepciones

Funciones

Estructuras de Datos

Misión TIC 2022

unab
VIGILADA MINEDUCACIÓN

Misión TIC 2022