El futuro digital es de todos

MinTIC

Mision TIC 2022

Semana 2

John Anderson Gómez Múnera

UNIVERSIDAD DE ANTIOQUIA
1803
Facultad de Ingeniería

Temas de la semana

Preguntas

Lógica booleana
Estructuras de decisión
Arreglos de Datos
Iteración en estructuras

Requisitos Funcionales

Álgebra Booleana

Operaciones:

- ‘y’ ‘and’
- ‘o’ ‘or’
- ‘no’ ‘not’

‘not’	
P	‘not’ P
VERDADERO	FALSO
FALSO	VERDADERO

‘and’		
P	Q	P ‘and’ Q
VERDADERO	VERDADERO	VERDADERO
VERDADERO	FALSO	FALSO
FALSO	VERDADERO	FALSO
FALSO	FALSO	FALSO

‘or’		
P	Q	P ‘or’ Q
VERDADERO	VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO
FALSO	VERDADERO	VERDADERO
FALSO	FALSO	FALSO

Recordar operadores

Se leen de izquierda a derecha:

>	: Mayor que
<	: Menor que
>=	: Mayor igual que
<=	: Menor igual que
!=	: Diferente
==	: Igual

Álgebra Booleana

¿ 5<2 ?

¿ (5<2) **or** (4>2) ?

¿ (5<2) **and** (4>2) ?

¿ **not** ((5<2) **and** (4>2)) ?

Álgebra Booleana

Evaluación de expresiones:

1. Expresiones aritméticas
2. Expresiones relacionales
3. Operadores lógicos

4*2+5 >= 30+2 or 5+3 != 35

Estructuras de datos

Son formas de agrupas muchos datos en un solo lugar

Estructura de datos ordenada que se pueden modificar

Listas

Estructuras de datos ordenadas que NO se pueden modificar

Tuplas

Estructuras de datos que representan conjunto matemáticos. No tienen orden

Sets

Estructuras desordenadas de pares de datos. Definidos de tal forma que a una llave le corresponde un dato

Diccionarios

Listas [ ]

Estructura de datos ordenada que se pueden modificar.
Pueden ser de diferentes tipos de datos.

```
my_list = ['a', 'c', 'Hello', 18, True]
longitud=len(lista)
lista[1]
lista[-2]
lista[1:4]
lista[4:]
lista[:-3])
```

Tuplas ()

Estructuras de datos ordenadas que NO se pueden modificar

```
numeros=(1,3,5,7,9,11)
vocales=('a','e','i','o','u’)
numeros[3]
numeros=numeros+(13,15,19)
```

Sets { }

Estructuras de datos que representan conjunto matemáticos. No tienen orden

```
con={1,2,"hola", True,2}
numeros={1,2,3,5,4}
frutas={"manzana", "peras","higos"}
tienda=numeros.union(frutas)
nuevo=tienda.intersection(frutas)
```

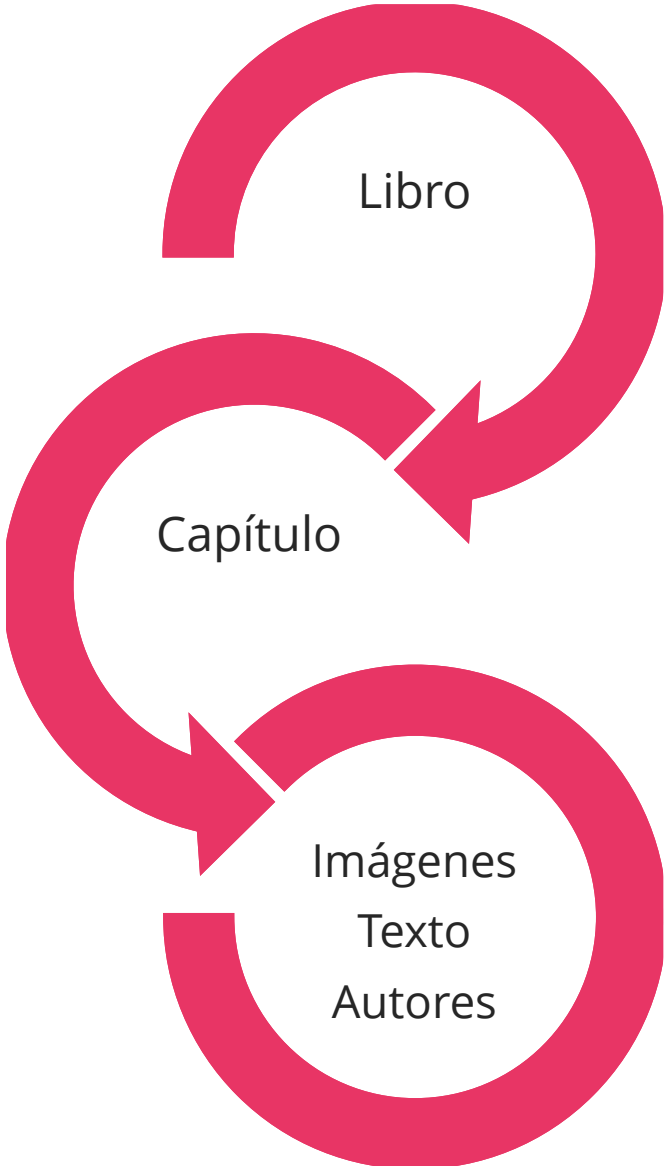
Diccionarios {“llave”: dato , }

Estructuras desordenadas de pares de datos. Definidos de tal forma que a una llave le corresponde un dato.

```
diccionario={
    "Peras“        :20,
    "manzanas“ : 2
}

diccionario["Peras"]
```

```
autos ={
  "marca“        : "mazda",
  "asegurado“  : False,
  "año“          : 1964,
  "colores“      : [“azul", “blanco", “rojo"]
}
```



Anidación

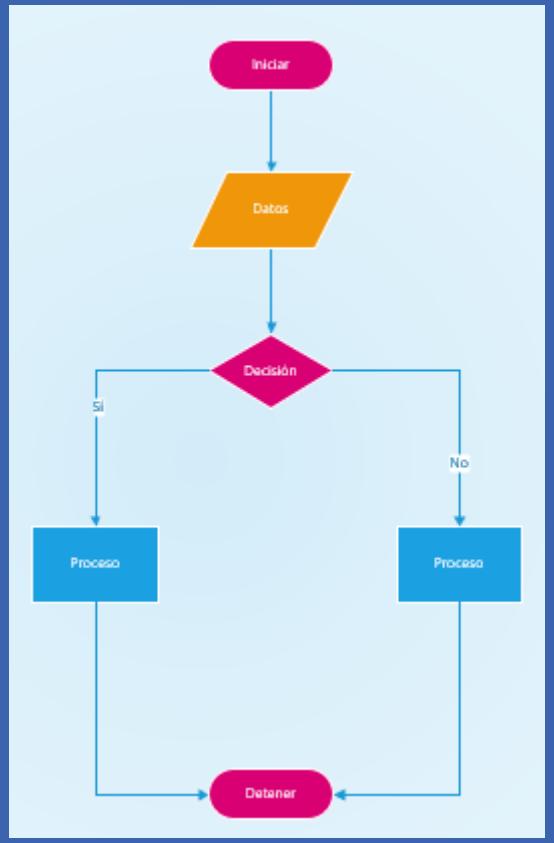
Anidación se refiere a poner una estructura de datos dentro de otra

FACULTAD DE INGENIERIA UdeA
Una facultad para la sociedad del aprendizaje

Estructuras de Control de Flujo

Una estructura de control, es un bloque de código que permite agrupar instrucciones de manera controlada. En Python es necesario tener en cuenta la identación.

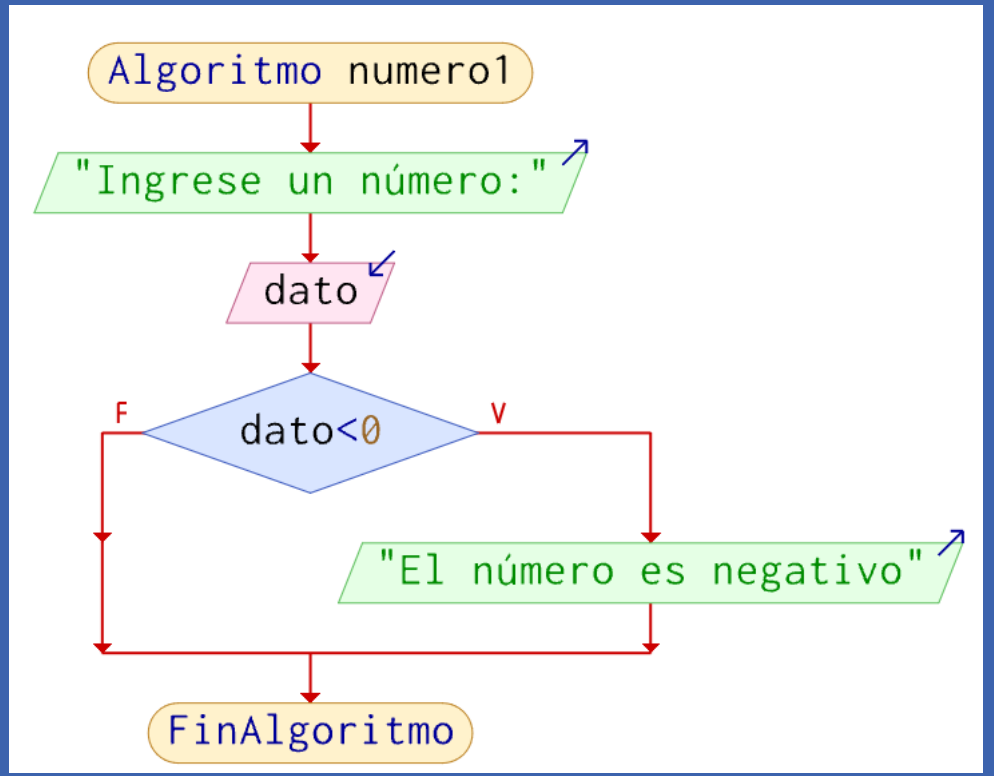
inicio de la estructura de control:
expresiones



Declaración if

```
if <Condición que se evalúa como booleana>:
        declaración
```

Ejemplo: Insertar un número y determinar si es negativo



Ejercicios if básicos:

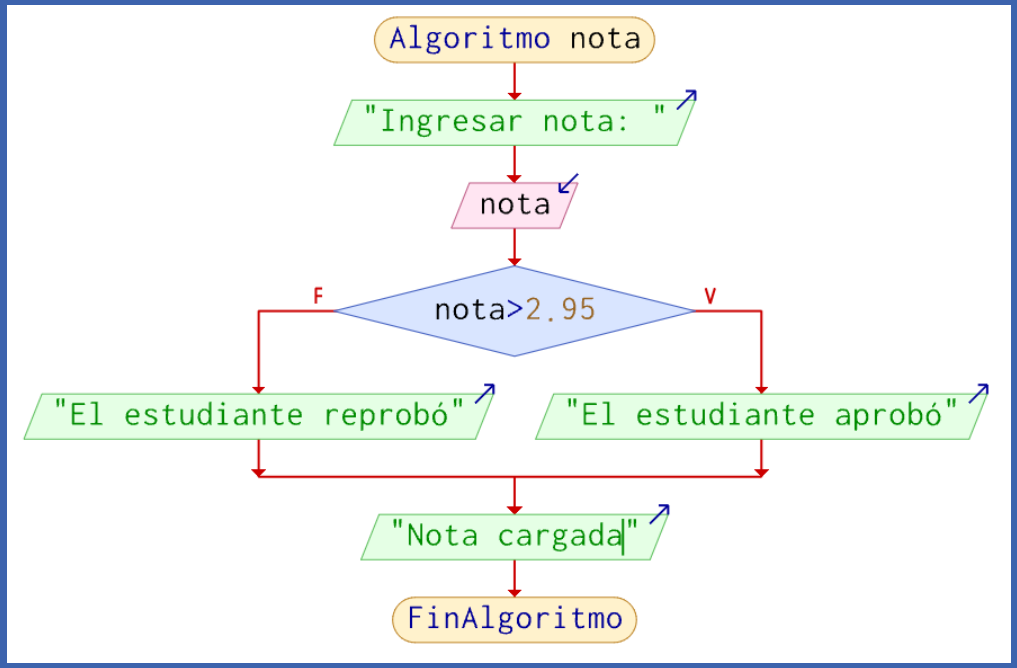
Ejercicio 1: Cree un programa donde solicite un número al usuario, si el número es positivo y menor que 10, se mostrará el mensaje "El número ingresado esta dentro del rango", al final del programa debe imprimir “Ejecución terminada"

Ejercicio 2: Cree un programa donde solicite un número al usuario, si el número es par, mostrando el mensaje "El número ingresado es par", al final del programa debe imprimir “Ejecución terminada"

Declaración if-else:

- La estructura if es necesaria para lograr cambiar el control de flujo de un programa (ofrecer varios caminos). Sin embargo, que pasa si se requiere que el programa deba tener en cuenta ambas alternativas: caso verdadero y caso falso.
- La sección “else” (en caso contrario) es una sección opcional que se puede añadir a la estructura “if”. Las instrucciones de esta sección se ejecutarán SOLO SI todas las condiciones anteriores se evaluaron como falsas

Ejemplo:



Ejercicios if-else:

Ejercicio 1: Cree un programa donde solicite un número al usuario, si el número correspondiente verifica que es mayor de edad, mostrar “Bienvenido a la fiesta”, sino mostrar el mensaje “Ingreso no permitido", al final del programa debe imprimir “Ejecución terminada"

Ejercicio 2: Cree un programa donde solicite un número al usuario y determine si es par o impar, al final del programa debe imprimir “Ejecución terminada"

Declaración if-elif-else:

Esta estructura permite ofrecer múltiples caminos dependiendo de múltiples comprobaciones

Ejemplo: Realizar un programa que ingrese el valor del gasto, si el gasto es hasta 50000$, pago con dinero en efectivo. Si gasto es más de 50000$ pero menos de 200000$, pago con tarjeta de débito. Sino, pago con tarjeta de crédito.

Ejercicio: Solicitar al usuario que ingrese el mes, con la primera palabra en mayúscula (ejemplo Mayo), y el programa imprima el mes y el número de días correspondiente.

Declaración if anidados:

Es posible contener if's dentro de otros if. Este término de anidación indica que una sentencia if se encuentra dentro de una parte de otra sentencia if y puede ser para refinar el comportamiento condicional del programa.

Ejemplo: utilice ciclo anidado para realizar un programa que permita al usuario ingresar el dinero disponible y la temperatura, si la temperatura es alta, además cuenta con dinero suficiente, comprar cerveza, si la temperatura es alta y no tiene dinero suficiente, imprimir comprar agua.

Con cuál operador lógico podría simplificar esto?

Ejercicio: Utilice ciclo anidado para realizar un programa en el cual el usuario ingrese 2 notas de un estudiante, si el promedio es menor a 2, imprimir que reprobó, si esta entre 2 y 3, imprimir "posibilidad de recuperatorio", y si es mayor a 3 colocar aprobó.

https://quizizz.com/join?gc=10757978

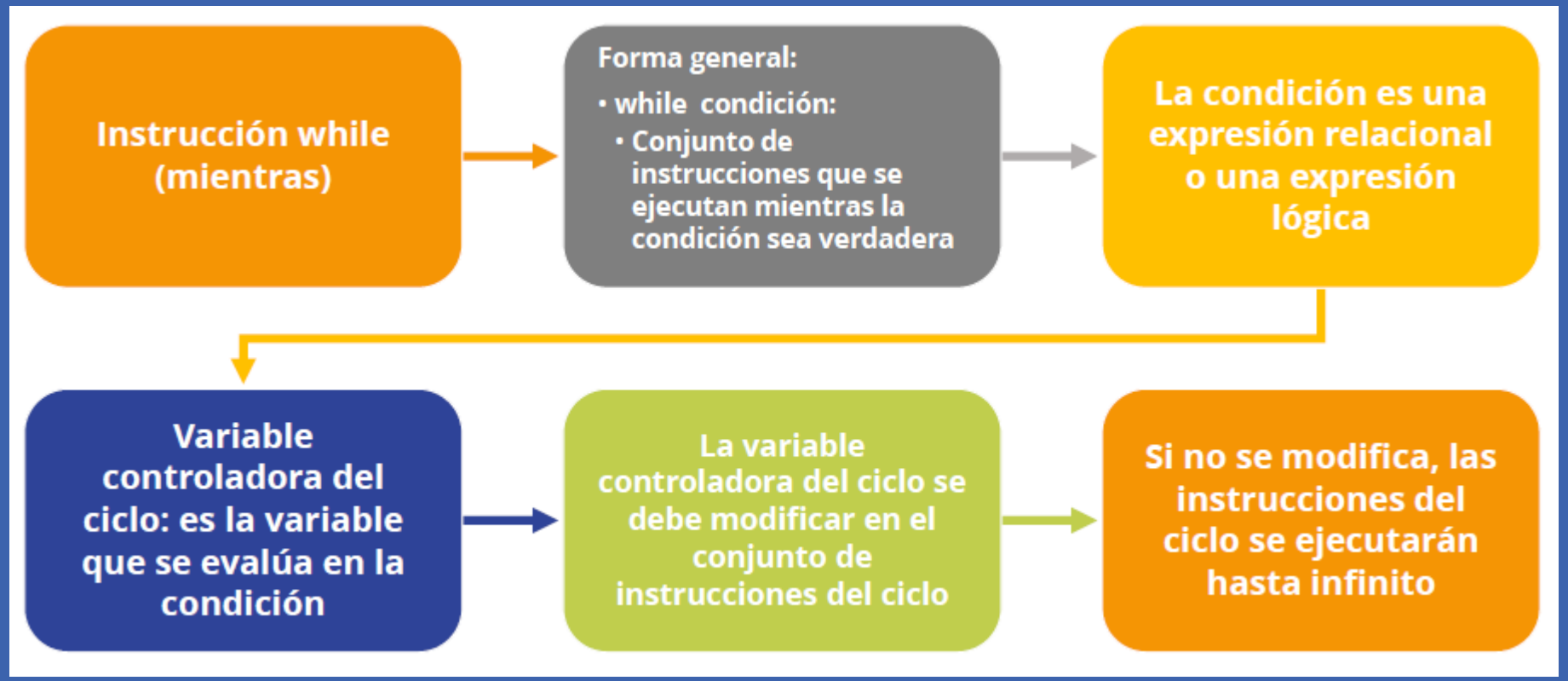
Estructuras de control iterativas

A diferencia de las estructuras de control condicionales, las iterativas (también llamadas cíclicas o bucles), nos permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición.

En Python se dispone de dos estructuras cíclicas:

- El bucle while
- El bucle for

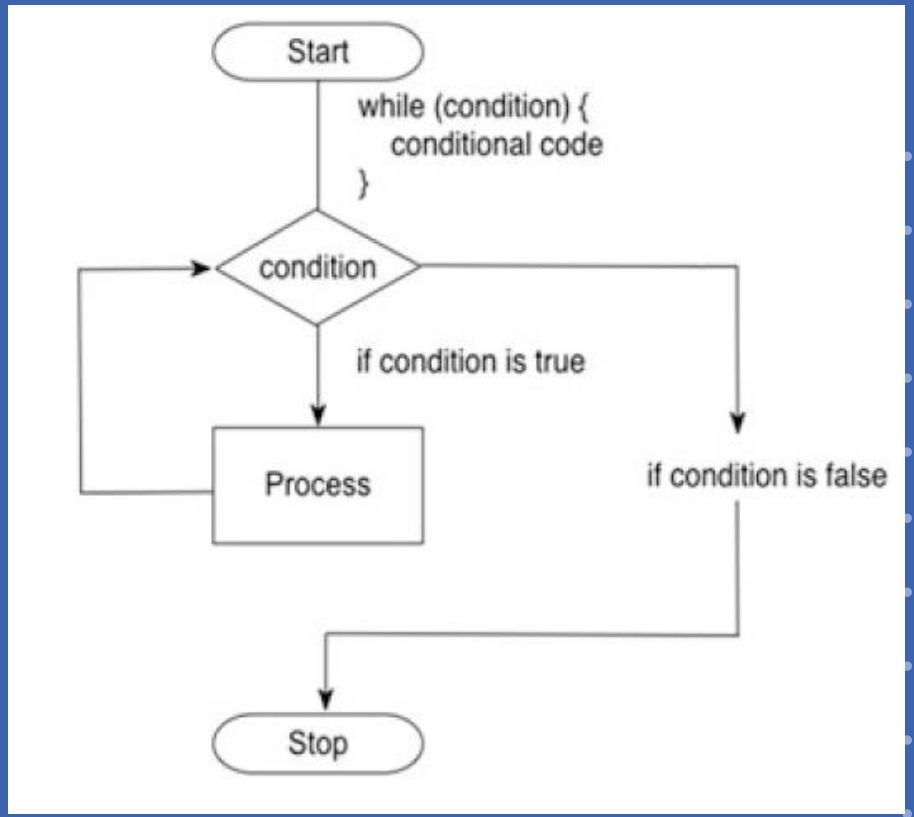
Instrucción while



Ciclo while

```
while <test-condition-is-true>:
statement or statements
```

La prueba se realiza antes de cada iteración, incluida la primera; Por lo tanto, si la condición falla en la primera vuelta del bucle, la sentencia o el bloque de sentencias puede no ejecutarse nunca.



Ejemplo:

```
count = 0
print('Empezando')
while count < 10:
        print(count, ' ', end='') # parte del ciclo while
        count += 1 # parte del ciclo
print() # no hace parte del ciclo
print('Hecho')
```

Ciclo while

Variables de control utilizadas

Contador: Cuenta el número de ocurrencias de un evento dentro de un ciclo.

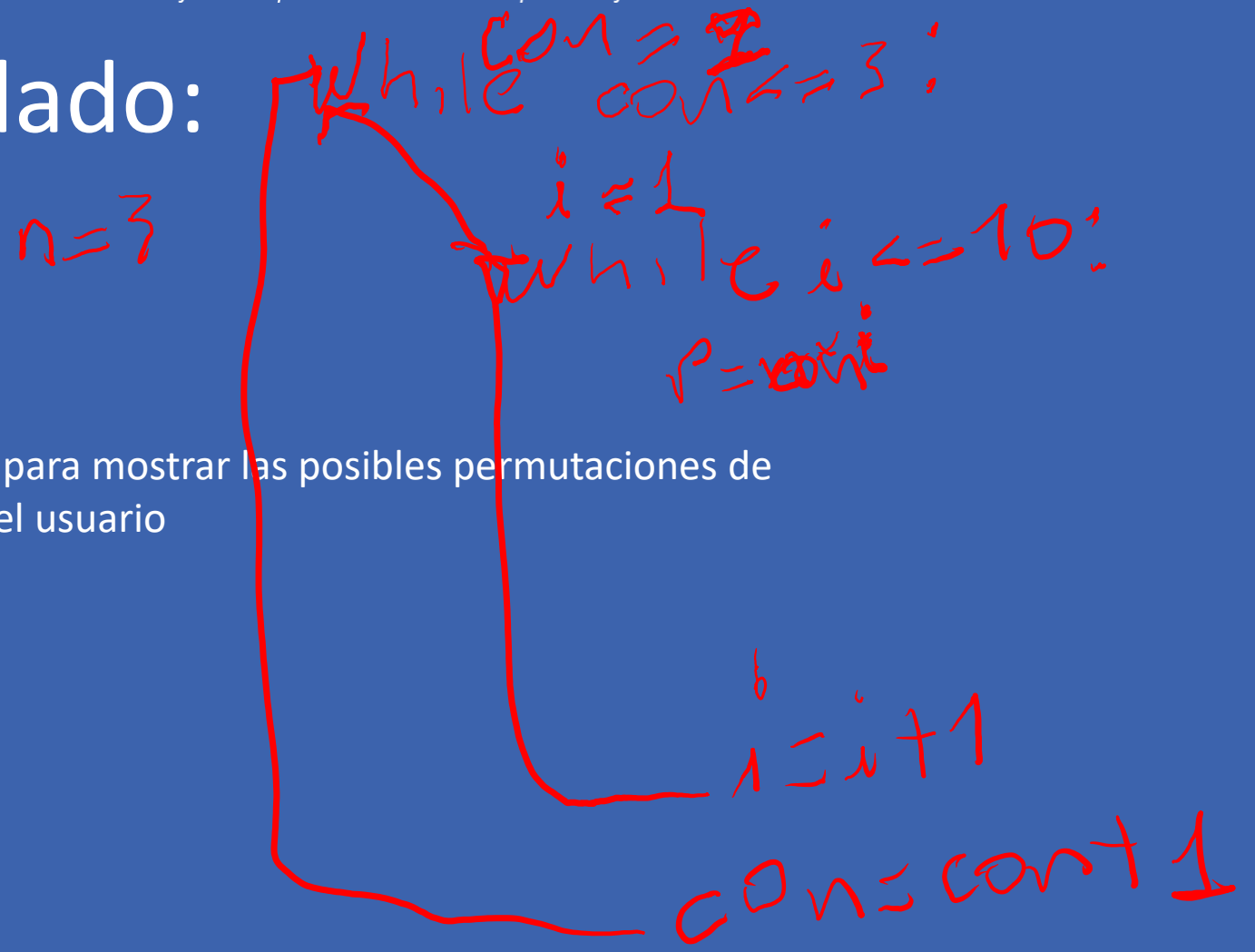
Acumulador: es una variable en la cual se lleva el total de un concepto específico en un ciclo.

Promedio: es el valor medio de un evento cuantificable. Se obtiene dividiendo un acumulador entre su respectivo contador.

Ejercicio: Un zoológico determina el precio de la entrada en función de la edad del visitante. Los visitantes de 2 años o menos son admitidos sin cargo. Los niños de entre 3 y 12 años cuestan 10,000$. Los mayores de 65 años cuestan 12,000$. La admisión para todos los demás invitados es de 20,000$. Cree un programa que comience leyendo las edades de todos los invitados de un grupo de usuarios, con una edad introducida en cada línea. El usuario introducirá una línea en blanco para indicar que no hay más visitantes en el grupo. El programa debe mostrar el costo de la entrada para el grupo con un mensaje apropiado con dos decimales, Además debe mostrarse el promedio.

Ciclo while anidado:

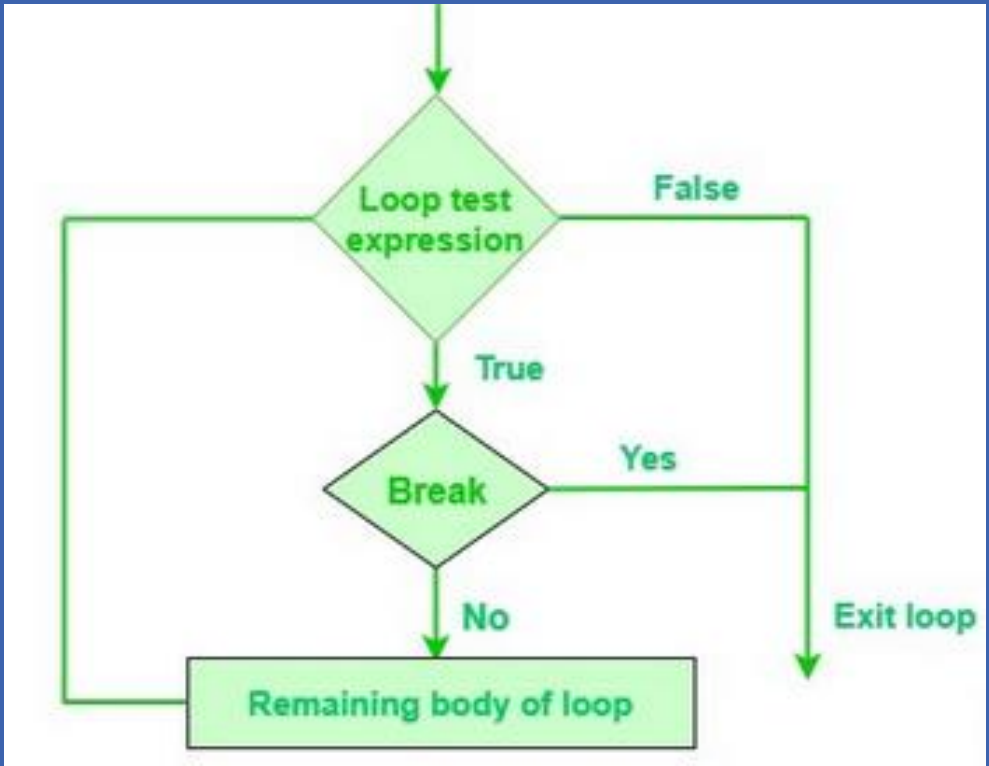
Ejemplo: Tablas de multiplicar



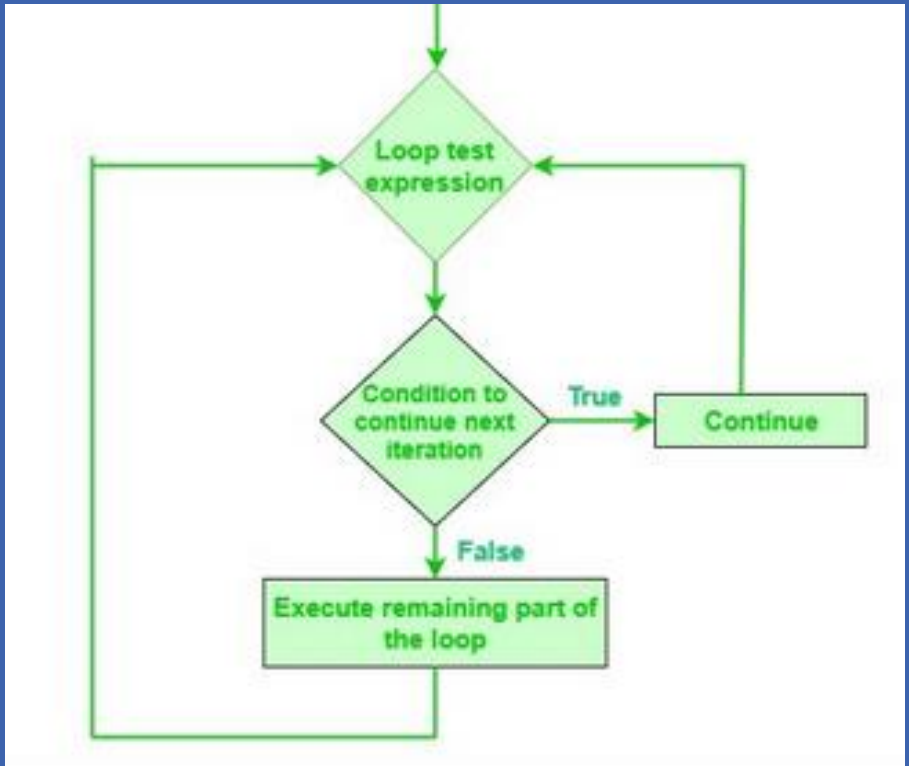
Ejercicio: Usar un ciclo while anidado para mostrar las posibles permutaciones de dos números enteros ingresados por el usuario

Declaraciones break, continue y pass

break



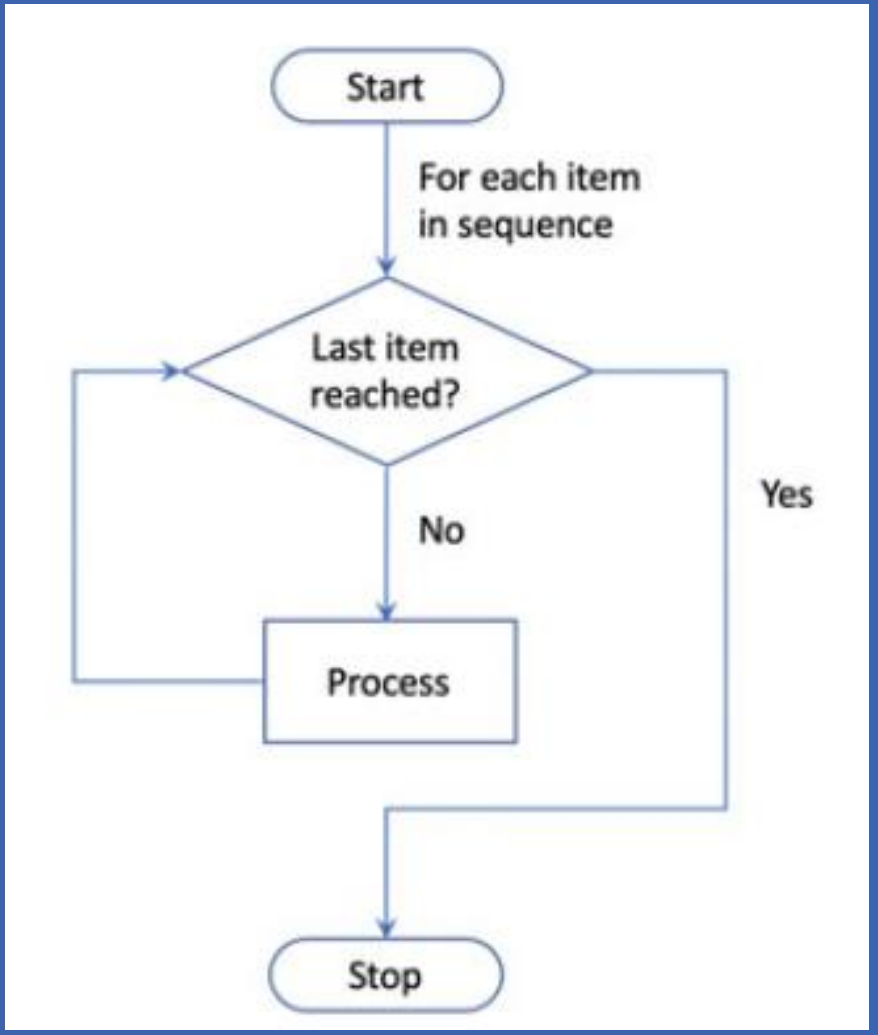
Continue



Ciclo For

En muchos casos sabemos cuántas veces queremos iterar sobre una o varias declaraciones. Aunque el bucle while puede utilizarse para estas situaciones, el bucle for es una forma mucho más concisa de hacerlo. También suele ser más claro para otro programador que el bucle debe iterar durante un número específico de iteraciones. número de iteraciones.

```
for <variable-name> in range(...):
    statement
    statement
```



Ciclo for

Ejemplo:

```
# Recorre un conjunto de valores en un rango
print('Imprime los valores en un rango')
for i in range(inicial, final):
    print(i, ' ', end='')
    print()
print('Hecho')
```

i: variable controladora del ciclo
Inicial: valor inicial (opcional)
Final: valor final (obligatorio)

Diferencias con while:

- El código es más conciso
- No necesita definir las variables primero

Ciclo for

Ejercicio: verificar si es un palíndromo. Una cadena es un palíndromo si es idéntica hacia adelante y hacia atrás. Por ejemplo, "anna", “salas", “otto" y "hannah" son ejemplos de palabras palindrómicas. Escriba un programa que lea una cadena del usuario y utilice un bucle para determinar si es o no un palíndromo. Muestra el resultado, incluyendo un mensaje de salida significativo.

https://quizizz.com/join?gc=27797338

Ciclo for

Uso de variable anónima

Una variación interesante del bucle for es el uso de un comodín (un '_') en lugar de una variable de bucle; esto puede ser útil si sólo se está interesado en hacer un bucle un cierto número de veces y no en el valor del contador del bucle en sí.

```
for i in range(10):
```

Instrucción break con for

```
num = int(input('Entre un número para verificar con el for: '))
for i in range(0, 6):
        if i == num:
                break
        print(i, ' ', end='')
print('Hecho')
```

Instrucción continue con for

```
for i in range(0, 10):
        print(i, ' ', end=‘’)
        if i % 2 == 1:
                continue
        print(‘Es un número par’)
print(‘Programa finalizado')
```

for con else

Un bucle for puede tener un bloque else opcional al final del bucle. La parte else se ejecuta si y sólo si se procesan todos los elementos de la secuencia. El bucle for no puede procesar todos los elementos del bucle si por alguna razón se produce un error en su programa (por ejemplo, si tiene un error de sintaxis) o si rompe el bucle.

https://quizizz.com/join?gc=24651610

Requisitos Funcionales

Define una función del sistema de software o sus componentes, conociendo los requisitos relevantes.

Pueden ser:

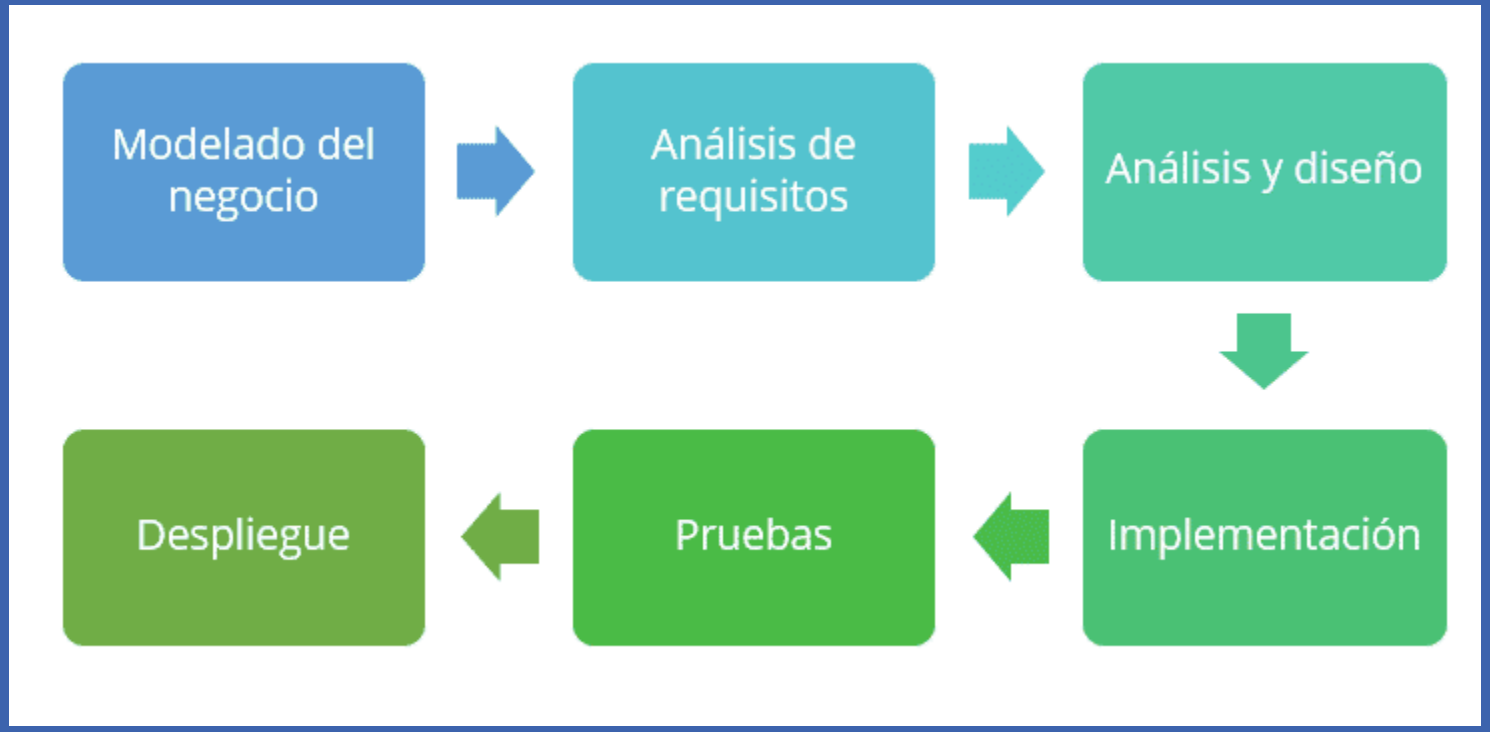
- Cálculos
- Detalles técnicos
- Manipulación de datos
- Tareas especificas que el programa debe cumplir

Requisitos Funcionales

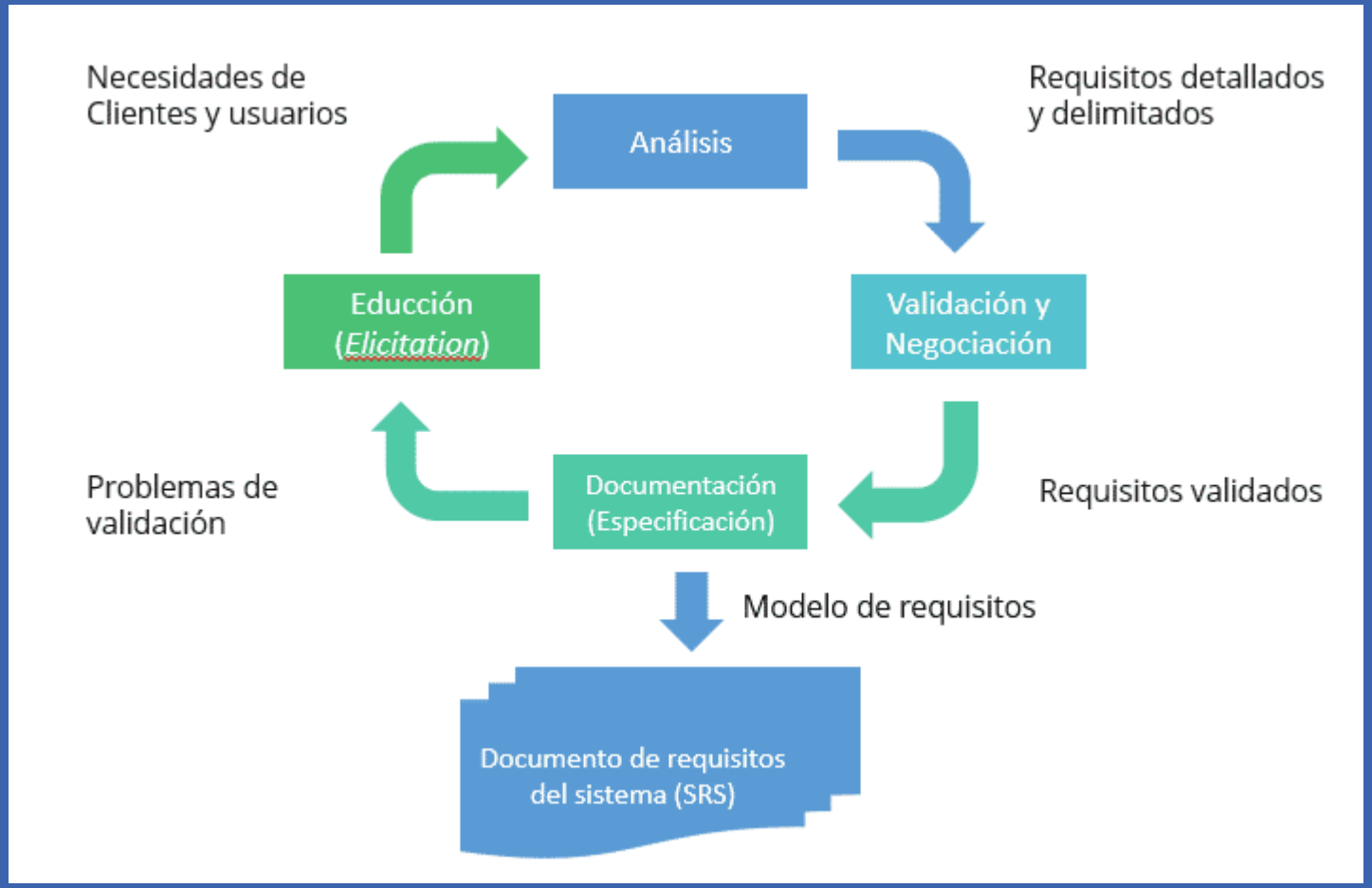
Acuerdo entre los implicados:

- Cliente
- Desarrollador

Ingeniería de Requisitos

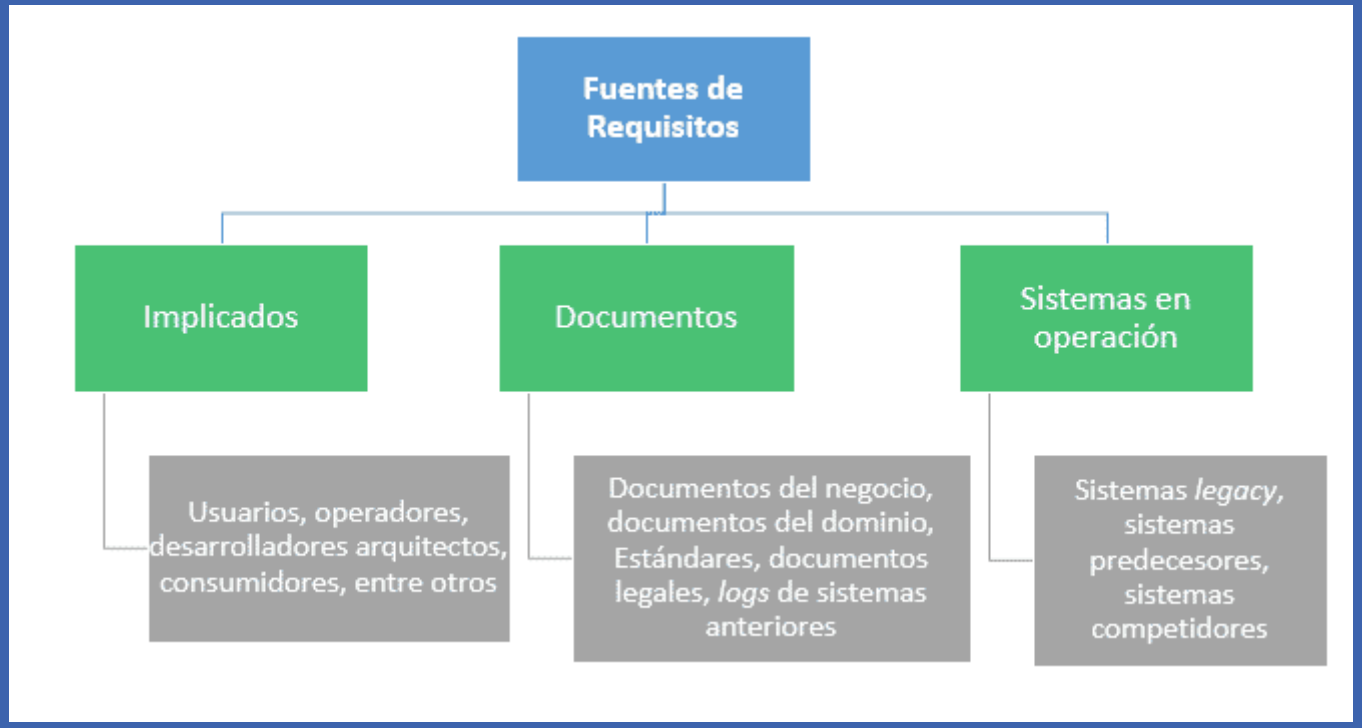


Gestión de requisitos



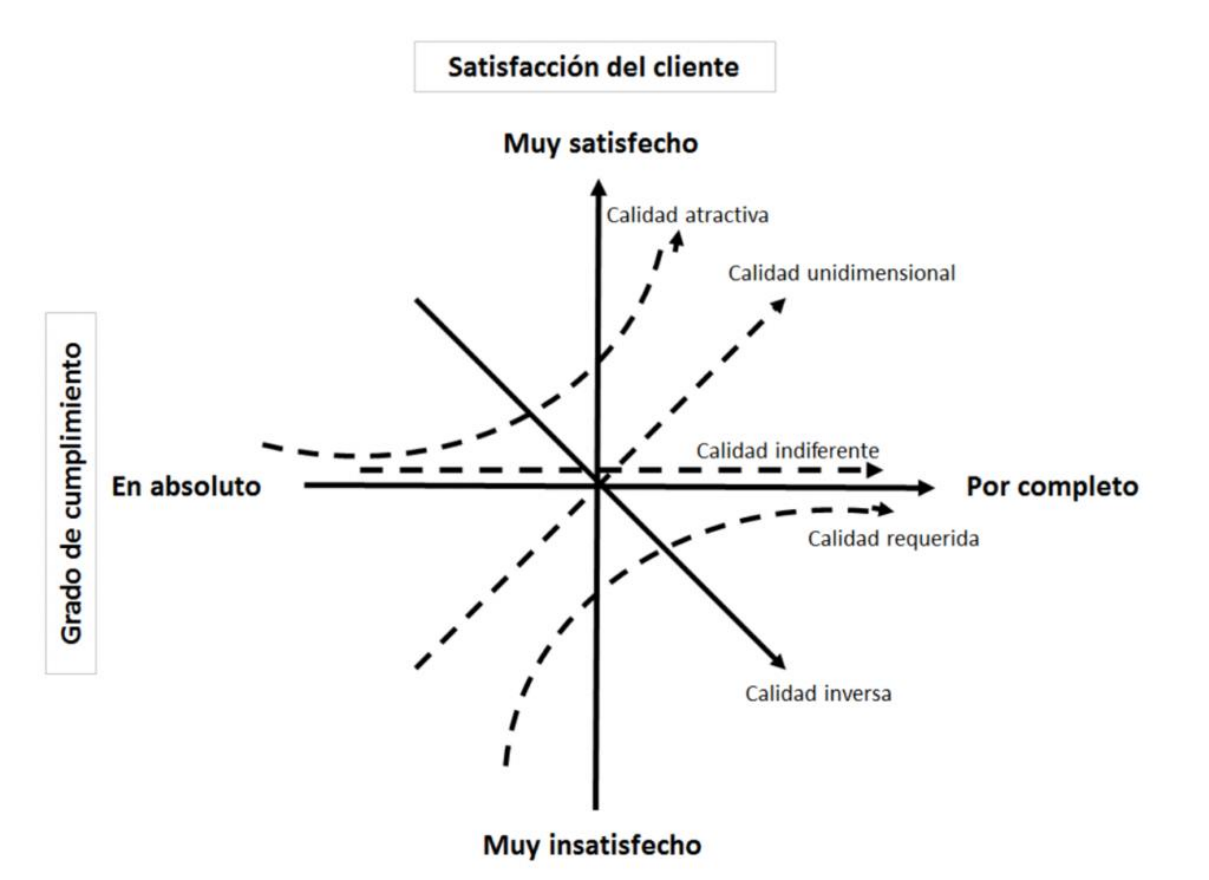
Educción de requisitos

Consistente en hallar o deducir los requisitos de un determinado sistema de información

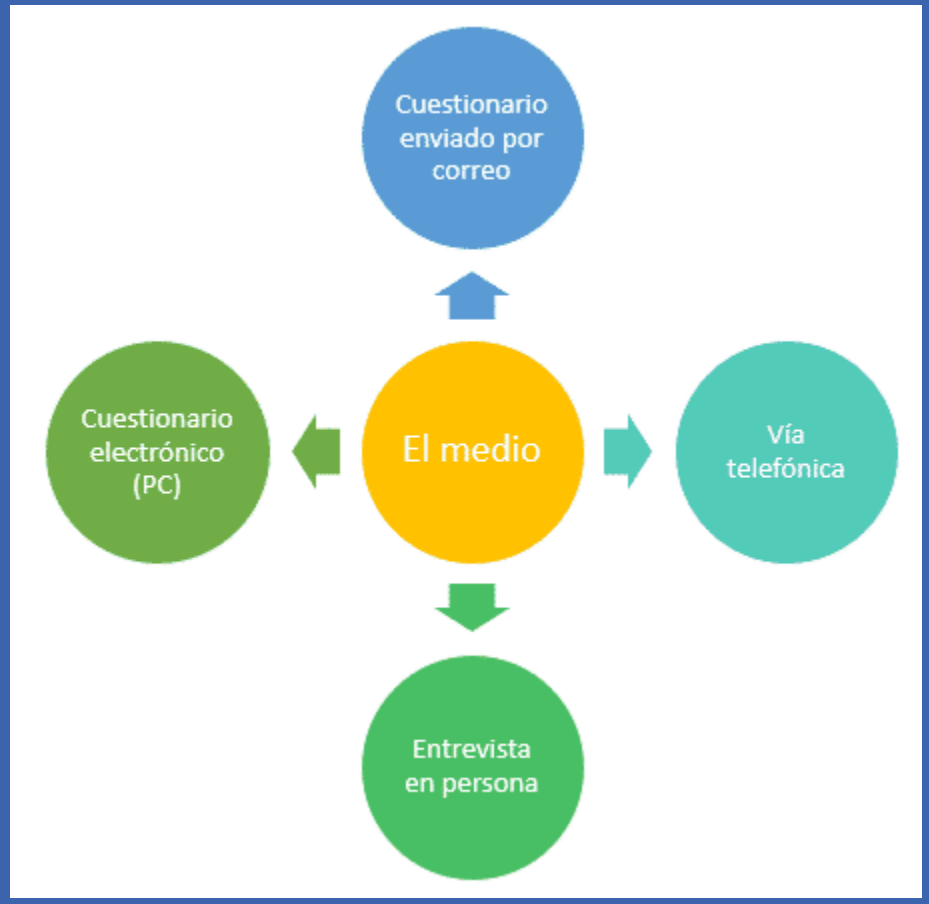


Modelo Kano

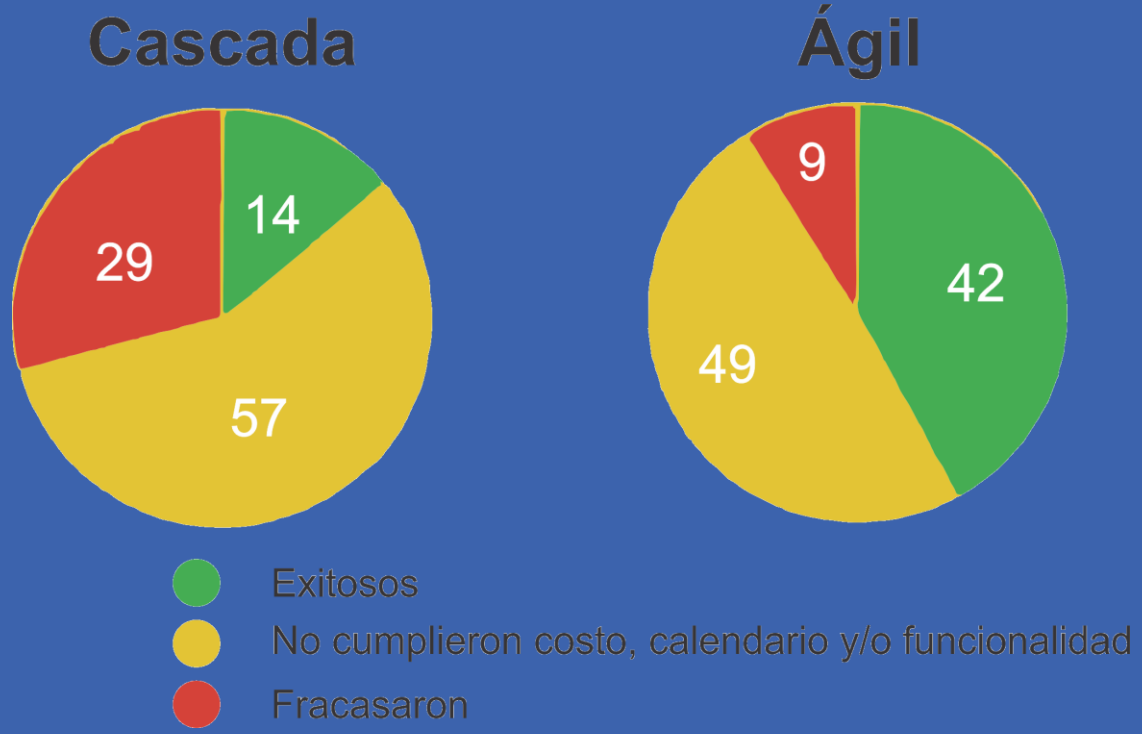
Teoría de desarrollo de productos y de satisfacción del cliente, basado en 5 categorías



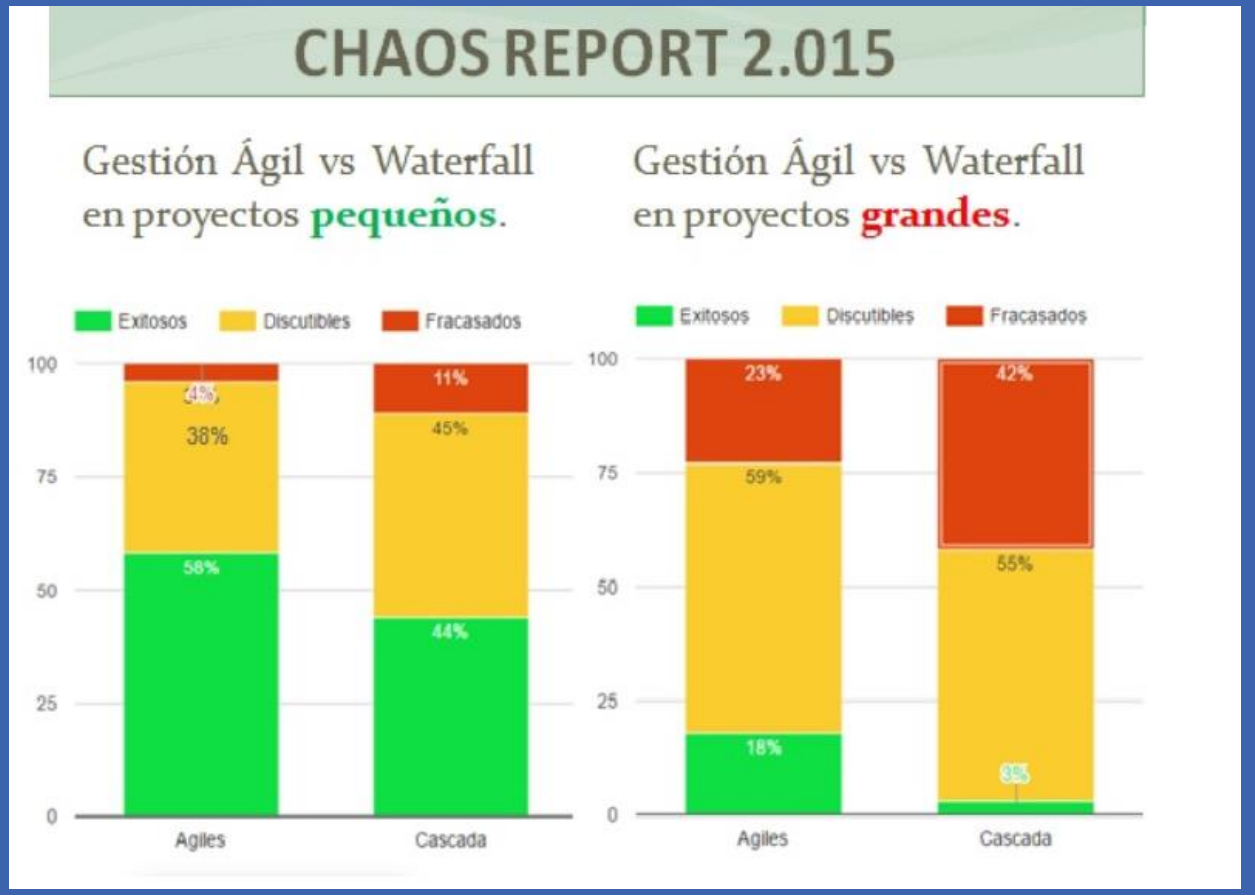
Métodos de recolección de requisitos



Metodologías Ágiles



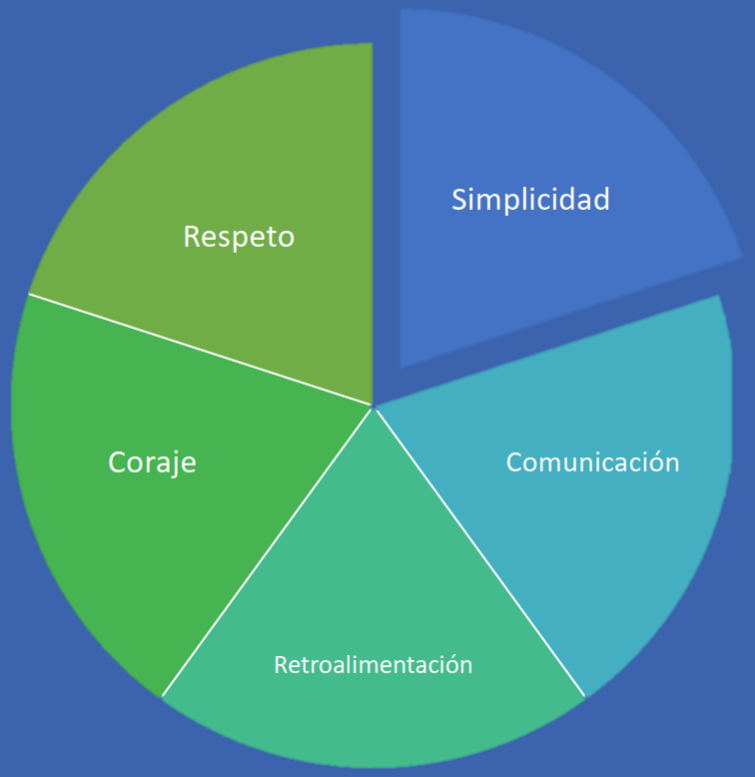
Metodologías Ágiles



Metodología Ágil

Metodología Ágil

eXtreme Programming



Simplicidad

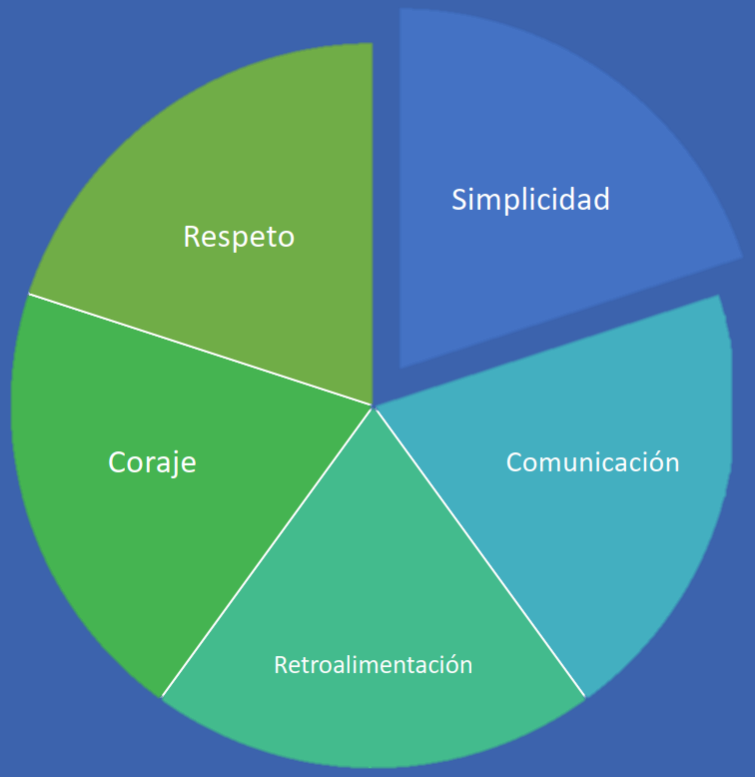
La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento.

Para mantener la simplicidad es necesaria la refactorización del código, esta es la manera de mantener el código simple a medida que crece.

También se aplica la simplicidad en la documentación.

Elegir adecuadamente los nombres de las variables, métodos y clases. Los nombres largos no decrementan la eficiencia del código ni el tiempo de desarrollo
Autoría colectiva del código y la programación por parejas

eXtreme Programming



Comunicación

El código autodocumentado es más fiable que los comentarios.

Debe comentarse solo aquello que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método.

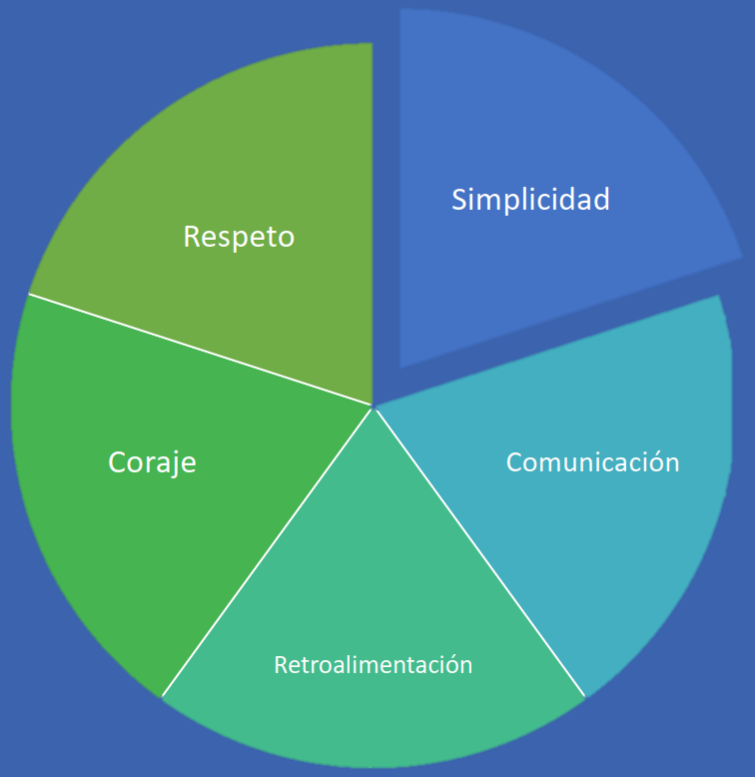
Las pruebas unitarias (Diseño de las clases, objetos y ejemplos).

Los programadores se comunican constantemente gracias a la programación por parejas.

La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo.

El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

eXtreme Programming



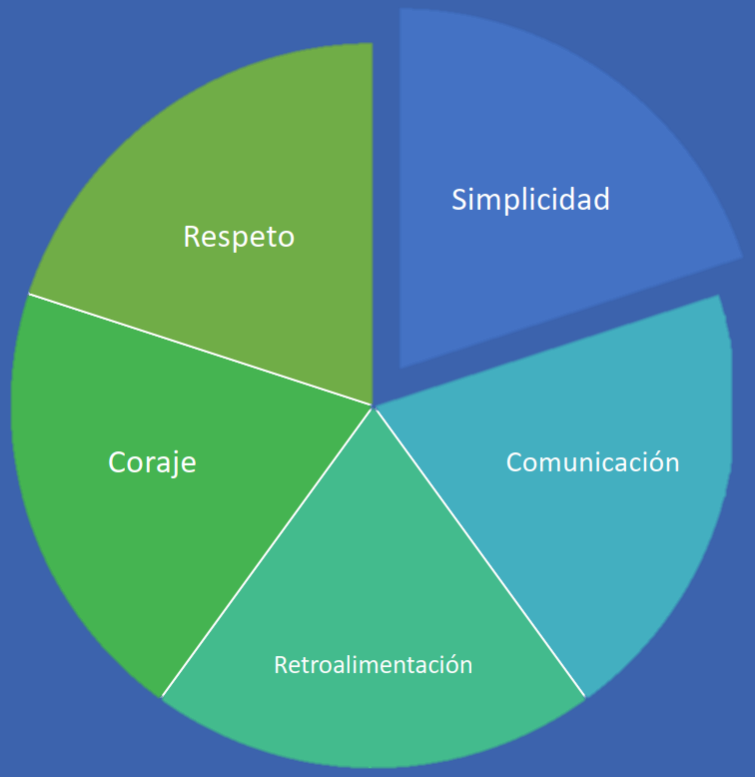
Retroalimentación

Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.

Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

eXtreme Programming



Coraje

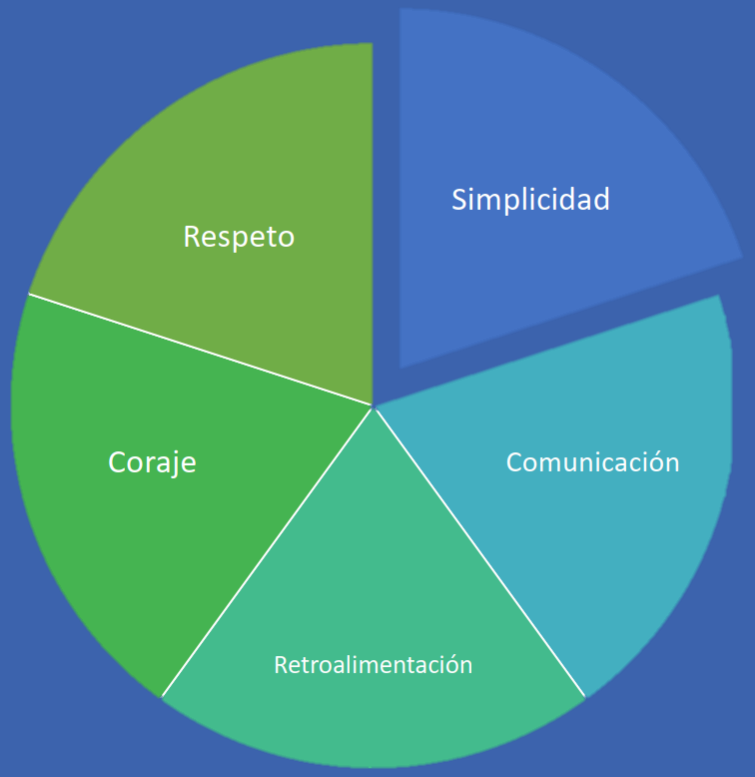
Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana.

La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario.

Valentía para quitar código fuente obsoleto, sin importar cuanto esfuerzo y tiempo se invirtió en crear ese código.

Valentía significa persistencia: un programador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, solo si es persistente.

eXtreme Programming



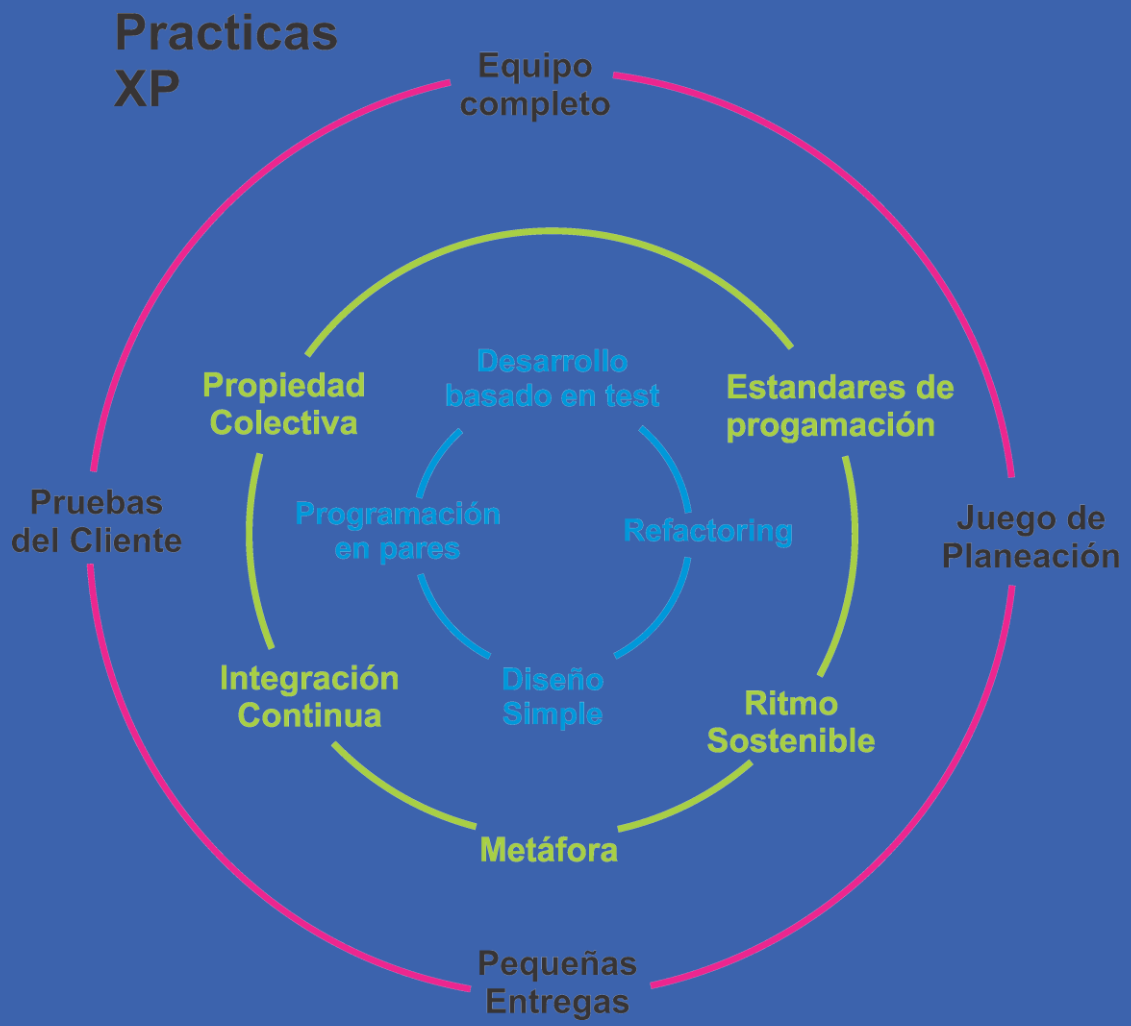
El Respeto

El respeto se manifiesta de varias formas.
Los miembros del equipo se respetan los unos a otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros.

Los miembros respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o más eficiente para la solución a través de la refactorización del código.

Los miembros del equipo respetan el trabajo del resto manteniendo las relaciones humanas cordiales

eXtreme Programming



eXtreme Programming