Conceptos de programación

En este documento se presentan los conceptos básicos de programación que se deben conocer para poder desarrollar software.

No están ligados a un lenguaje de programación en específico, sino que son conceptos generales que se pueden aplicar en cualquier lenguaje.

◇

Proceso computacional

Ente abstracto que puede realizar cálculos y toma de decisiones en base a un conjunto de instrucciones.

Utiliza `datos` como entrada, `instrucciones` para procesarlos y `resultados` como salida.

Datos

`Representación simbólica` de hechos, conceptos o instrucciones en el formato `adecuado` para su procesamiento.

Pueden ser números, texto, imágenes, sonidos, etc.

Algoritmo

Conjunto finito de instrucciones que `describen` un `proceso computacional`.

Una receta que describe paso a paso cómo resolver un problema.

Programa

Implementación de un algoritmo en un `lenguaje de programación`.

Cada uno seguirá un formato específico y estará compuesto por `instrucciones` y `datos`.

Lenguaje de programación

Conjunto de `símbolos` y `reglas` que permiten escribir programas.

Cada lenguaje tiene características y `funcionalidades` específicas.

◇

Palabras reservadas

Símbolos que tienen un `significado` específico en un lenguaje de programación.

NO pueden ser usadas como nombres de variables o funciones.

Python:

```
if = 1 # Error
```

Javascript:

```
for = 1 // Error
```

Variables

`Espacios de memoria` que se utilizan para almacenar `valores`.

Pueden tomar diferentes `tipos de datos` y ser `modificadas` durante la ejecución del programa.

```
gato = "Michi"
edad = 3
```

Valor

`Dato` almacenado en una variable.

Referencia

`Dirección de memoria` donde se encuentra almacenado un valor.

Tipos de datos

Formas en que se pueden representar los valores en un lenguaje de programación.

Normalmente se dividen en `primitivos` y `compuestos`.

Mutabilidad e inmutabilidad

Algunos tipos de datos pueden ser modificados después de su creación, mientras que otros no.

```
lista = [1, 2, 3]
lista.append(4)
print(lista) # [1, 2, 3, 4]
tuple = (1, 2, 3)
tuple.append(4) # Error
```

◇

Tipado fuerte y débil

Algunos lenguajes de programación permiten realizar operaciones entre diferentes tipos de datos, mientras que otros no.

Python

```python
numero = 1
texto = "2"
print(numero + texto) # Error
```

Javascript:

```javascript
let numero = 1
let texto = "2"
console.log(numero + texto) // 12
```

Tipado estático y dinámico

Algunos lenguajes de programación requieren que las variables tengan un tipo de dato específico, mientras que otros no.

Primitivos

Algunos de los tipos de datos primitivos más comunes son:

- Enteros
- Flotantes
- Cadenas de texto
- Booleanos
- Nulos
- Bytes

Enteros

Números enteros, por ejemplo: 1, 312, -3, 0

Nos sirven para representar cantidades enteras, como la cantidad de elementos en una lista, la cantidad de veces que se repite un ciclo o la cantidad de vidas de un personaje en un videojuego.

Por lo general, se representan con el tipo `int`.

Flotantes

Números decimales, por ejemplo: 3.14, -0.5, 1.0

Nos sirven para representar cantidades que pueden tener decimales, como la velocidad de un objeto, la posición de un personaje en un videojuego o el precio de un producto.

Por lo general, se representan con el tipo `float`.

Cadenas de texto

Secuencias de caracteres, por ejemplo: "Hola mundo", "Python" o el contenido de un .txt.

Nos sirven para representar texto, como mensajes, nombres, direcciones, etc.

Por lo general, se representan con el tipo `str`.

Booleanos

Valores de verdad, por ejemplo: True, False

Nos sirven para representar condiciones, como si un objeto está activo o inactivo, si un usuario está logueado o no, si un personaje está vivo o muerto, etc.

Por lo general, se representan con el tipo `bool`.

Nulos

Valores nulos, por ejemplo: None

Nos sirven para representar la ausencia de un valor, como cuando una variable no ha sido inicializada o cuando un objeto no ha sido creado.

Es un dato que no tiene valor, pero que puede ser útil para indicar que algo no está definido, según el lenguaje de programación y el contexto, su comportamiento y resultados pueden variar.

Compuestos

Datos compuestos por otros datos, primitivos o compuestos.

Listas o arrays

Colecciones ordenadas mutables de elementos.

Pueden contener cualquier tipo de dato, primitivo o compuesto.

Se acceden por su posición o `índice`.

```python
#Python
nombres = ["Juan", "Pedro", "María"]
print(nombres[0]) # Juan
```

```java
//Java
String[] nombres = {"Juan", "Pedro", "María"};
System.out.println(nombres[0]); // Juan
```

```javascript
//Javascript
let nombres = ["Juan", "Pedro", "María"];
console.log(nombres[0]); // Juan
```

Diccionarios o mapas

Colecciones no ordenadas de elementos, compuestos por una `clave` y un `valor`.

Se acceden por su clave.

```python
#Python
usuario = {
    "nombre": "Pepito",
    "edad": 30,
    "estado": "Activo"
}

print(usuario["nombre"]) # Pepito
```

Python:

```python
velocidad = 30
mensaje = "Hola mundo"
nombres = ["Juan", "Pedro", "María"]
```

Java:

```java
int velocidad = 30;
String mensaje = "Hola mundo";
String[] nombres = {"Juan", "Pedro", "María"};
```

◇

Operadores

Símbolos que permiten realizar `operaciones` entre `valores y variables`.

Algunos de los operadores más comunes son:

- Aritméticos
- Lógicos
- Relacionales
- Asignación

Operadores aritméticos

Nos permiten realizar operaciones matemáticas entre valores.

```
a = 10
b = 3

print(a + b) # 13
print(a - b) # 7
print(a * b) # 30
print(a / b) # 3.3333333333333335
print(a % b) # 1
print(a // b) # 3
```

También podemos realizar operaciones aritméticas con otros tipos de datos como cadenas de texto.

```
a = "Hola"
b = "mundo"
c = "!"

print(a + " " + b + (c * 3)) # Hola mundo!!!
```

Operadores lógicos

Nos permiten realizar operaciones lógicas entre valores booleanos.

```
a = True
b = False

print(a and b) # False
print(a or b) # True
print(not a) # False
print(not b) # True
print(a ^ b) # True
print(a ^ True) # False
```

Cuando los usamos con otros tipos de datos, los operadores lógicos pueden tener un comportamiento diferente según el lenguaje de programación.

```
a = "Hola"
b = "mundo"

print(a and b) # mundo
print(a or b) # Hola
print(not a) # False
```

Operadores de comparación

Nos permiten comparar dos valores y determinar si son iguales, diferentes, mayores, menores, etc.

```
a = 10
b = 3

print(a == b) # False
print(a != b) # True
print(a > b) # True
print(a < b) # False
print(a >= b) # True
print(a <= b) # False
```

Siempre devuelven un valor booleano.

Operadores de asignación

Nos permiten asignar un valor a una variable.

```
a = 10
b = 3

a += b
print(a) # 13
```

◇

Control de flujo

Mecanismos que permiten modificar el `flujo de ejecución` de un programa, es decir, la secuencia en la que se ejecutan las instrucciones.

- Secuenciales
- Condicionales
- Iterativas
- Transferencia de control
- Funciones
- Recursividad
- Excepciones
- Paralelismo
- Concurrencia

Secuenciales

Instrucciones que se ejecutan de arriba hacia abajo, en el orden en que aparecen en el programa.

```
print("Hola")
print("mundo")
```

◇

Condicionales

Instrucciones que se ejecutan `si` se cumple una `condición`.

```
a = 10

if a > 5:
    print("a es mayor que 5")
else:
    print("a es menor o igual que 5")
```

Siempre se evalúa una condición y se espera un valor booleano o un valor que pueda ser convertido a booleano.

Iterativas

Instrucciones que se ejecutan `mientras` se cumpla una `condición`.

```
a = 0

while a < 5:
    print(a)
    a += 1
```

Transferencia de control

Instrucciones que permiten `saltar` a una `parte específica` del programa.

```
for x in range(10):
    if x == 5:
        break
    print(x)
```

Funciones

Conjunto de instrucciones que realizan una tarea específica.

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

saludar("Juan")
```

Recursividad

Técnica que consiste en que una función se llame a sí misma.

```
def contar(n):
    if n == 0:
        return
    print(n)
    contar(n - 1)

contar(5)
```

Excepciones

Mecanismo que permite manejar errores y excepciones en un programa.

```
val1 = "un string"
val2 = 10

try:
    resultado = val1 + val2
except TypeError:
    print("No se pueden sumar un string y un entero")
```

Paralelismo

Ejecución de varias tareas de forma simultánea.

```
import threading

def scan():
    print("Escaneando red...")
    # Código para escanear red

def download():
    print("Descargando archivo...")
    # Código para descargar archivo

thread1 = threading.Thread(target=scan)
thread2 = threading.Thread(target=download)
```

Concurrencia

Ejecución de varias tareas de forma alternada.

```
import asyncio

async def scan():
    print("Escaneando red...")
    # Código para escanear red

async def download():
    print("Descargando archivo...")
    # Código para descargar archivo

asyncio.run(scan())
asyncio.run(download())
```

Funciones

Bloques de código que se pueden reutilizar en diferentes partes de un programa.

Las funciones pueden recibir uno o varios `parámetros` y `devolver` uno o varios `valores`.

```
def saludo(nombre):
    return "Hola " + nombre
```

Programación orientada a objetos

Paradigma de programación que se basa en la creación de `objetos` que contienen `atributos` y `métodos`.

Permite modelar el mundo real de una forma más sencilla y estructurada.

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        return f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años"
```

Librerias o modulos

Conjunto de funciones y objetos que se pueden importar en un programa.

Pueden ser `nativas` o `externas`.

```
import random

print(random.randint(1, 10))
```

```
import Flask

app = Flask(__name__)
```

Repositorios

Lugar donde se almacenan los archivos de un proyecto y un historial de los cambios del mismo.

No son sinónimo de `código fuente`, ya que pueden contener otros archivos como documentación, imágenes, etc.