

# Pseudocódigo

## Sintaxis básica

Durante la materia vamos a usar **pseudocódigo** o **C** para resolver problemas en exámenes y en clase. Nos va a interesar el **orden temporal** de las funciones que usemos para resolver de la forma más eficiente lo pedido.

## Tipos de datos

Para declarar una variable vamos a indicar el tipo precediendo al nombre de la variable.

```
<tipo> nombre = valor
```

Los tipos disponibles son:

**Numérico:** Un número con o sin decimales

**String:** Una cadena de caracteres. Se puede operar con ella como en Python.

**Booleano**

**Tipos de datos abstractos:** Otros tipos de datos

## Funciones

Para definir una función escribimos una firma donde indicamos que tipo devuelve la función y que tipos recibe, como lo haríamos en C.

```
numeric producto(numeric a, numeric b){  
    return a*b  
}
```

Podemos asumir que tenemos una batería de funciones disponibles:

Nombre	Descripción		Orden temporal
<i>sort</i>	Ordena arreglo/string	un	$O(n \log(n))$
<i>reverse</i>	Invierte arreglo/string	un	$O(n)$
<i>max/min</i>	Obtiene el máximo o mínimo de un arreglo		$O(n)$

Puede asumir que tiene disponible también funciones para operaciones matemáticas (*sqrt*, *cos*, ...) y preguntar en caso de necesitar otra no listada.

# Pseudocódigo

## Sintaxis básica

### Condición IF

Las condiciones IF tienen el siguiente formato:

```
if (promedio > 4){  
    return TRUE  
}
```

### Ciclos

Los ciclos definidos pueden expresarse de dos formas:

```
for i in [1,2,3,4]{  
    return TRUE  
}
```

```
for i in 1..4{  
    return TRUE  
}
```

Incluyendo en la iteración al primer y último elemento.

Los ciclos indefinidos se expresan de la siguiente forma:

```
while (continuar){  
    ...  
}
```

### Ejemplo

Se pide una función que determine si un numero es primo.

```
boolean es_primo(numeric numero){  
    for i in 2..floor(sqrt(numero)){  
        if (numero%i == 0){  
            return FALSE  
        }  
    }  
    return TRUE  
}
```

# Pseudocódigo

## Tipos de datos abstractos

### Listas

```
lista crear_lista()
```

```
insertar_ultimo(lista lista, elemento)
```

```
insertar_primerio(lista lista, elemento)
```

```
.. extraer_primerio(lista lista)
```

```
.. extraer_primerio(lista lista)
```

```
numeric cantidad(lista lista)
```

$O(1)$

### Colas

```
cola crear_cola()
```

```
encolar(cola cola, elemento)
```

```
.. desencolar(cola cola)
```

```
numeric cantidad(cola cola)
```

$O(1)$

### Pilas

```
pila crear_pila()
```

```
apilar(pila pila, elemento)
```

```
.. desapilar(pila pila)
```

```
numeric cantidad(pila pila)
```

$O(1)$

# Pseudocódigo

## Tipos de datos abstractos

### Heaps

```
heap crear_heap()
```

```
numeric cantidad(lista lista)
```

```
encolar(heap heap, elemento)
```

```
desencolar(heap heap, elemento)
```

}  $O(1)$

}  $O(\log(n))$

### ABBs

```
abb crear_abb()
```

```
numeric cantidad(abb arbol)
```

```
guardar(abb arbol, clave, elemento)
```

```
borrar(abb arbol, clave)
```

```
.. obtener(abb arbol, clave)
```

```
boolean pertenece(abb arbol, clave)
```

}  $O(1)$

}  $O(\log(n))$

### Hash

```
hash crear_hash()
```

```
numeric cantidad(hash hash)
```

```
guardar(hash hash, clave, elemento)
```

```
borrar(hash hash, clave)
```

```
.. obtener(hash hash, clave)
```

```
boolean pertenece(hash hash, clave)
```

}  $O(1)$