

2022/01

# Lenguajes y Paradigmas de Programación

# Procedural Programming

## Tarea 1

Como ayuda para que puedan instalar

- Instalar Python 3.10.2 (Disponible para Windows y MAC en [Python Download](#))
- Instalar Java 11 JDK (Disponible para Windows, MAC y Linux en [Java Download](#))
- Instalar GraalVM ([GraalVM Website](#))
- Instalar GIT ([Instrucciones en GIT Installation](#))
- Crear cuenta en GitHub ([GitHub Website](#))
- Instalar client GitHub ([GitHub Desktop](#))
- Instalar IDE de su preferencia ([Visual Studio Code](#) es bueno si no tienen alguno)

# Características

Imperativo

Estructurado

Basado en procedimientos

→ Modularidad, compartir código, fácil  
lectura

Asignación explícita

Discusión

# Juego en Python

La madre de todos los lenguajes y el más usado en la actualidad

# C y Python

# Componentes de código

## C

- Directivas de compilación
- Declaraciones
- Definición de variables (globales)
- Definición de métodos

## *Python*

- Directivas de “compilación” (Python no se compila per se)
- Definición de variables (globales)
- Definición de métodos

# Ejemplos de código en C

- Directivas de compilación

```
#define MAX 100  
#include <stdio.h>
```

- Declaraciones

```
void push( char c );
```

- Variables (globales)

```
int top; char Store[MAX];
```


- Métodos

```
void push( char c ) { ... }
```



# Ejemplos de código en C

```
#define MAX 100
#include <stdio.h>
void push( char c );
int top;
char store[MAX];
void push( char c ) {
    if (top < MAX)
        store[top++] = c;
}
```



Usualmente esto se conoce  
como el encabezado

Y esto como el cuerpo

# En Python

```
import numpy
MAX = 100
top = 0
store = numpy.zeros(MAX)
def push( c ):
    if (top < MAX):
        store[top+1] = c
        top = top + 1
push(1)
```

En Python no existe una diferencia clara  
como en C  
Y esto puede provocar problemas

# Ejemplos de código

En C, si el archivo .c contiene un método llamado **main** entonces puede ser ejecutado. De lo contrario se puede ocupar como una biblioteca (veremos eso después).

En Python todo código se ejecuta a menos que esté en una estructura de control

# Cuidado en Python

```
Traceback (most recent call last):  
  File "python", line 11, in <module>  
    File "python", line 7, in push  
UnboundLocalError: local variable 'top' referenced before assignment
```

Las variables globales en Python se pueden leer en métodos, pero no sobrescribir, A MENOS que se declare explícitamente eso

```
def push( c ):  
    global top  
    if (top < MAX):  
        store[top+1] = c  
        top = top + 1
```

Discutamos

**¿Por qué esa restricción con las variables globales?**

# Variables y tipos de datos

En C: Variable <tipo de datos> <nombre> (= <valor>);

Tipos de datos:

- Básicos: int, char, float
- “versiones” de estos tipos: short, long, double, long double, ...
- No existe el tipo boolean en C:
  - 0 => falso
  - no cero => true

En Python, no definimos tipo, el intérprete lo extrae.... Y si existen los booleanos (True/False)

# Operadores en C

Aritméticos: + - \* / %

Agrupación: ( )

Incremento: ++ --

Asignación: = += -= \*= /= %=

Lógicos: && || !

Otros ...

En Python, no hay operadores equivalentes. Debe usarse una forma como la de abajo: el equivalente de `i++` en C es `i+=1` en Python... y no directamente

En Python, los operadores lógicos equivalente son `and`, `or` y `not`

# Entrada / Salida

Salida: printf

- `printf("<formato>"(, <parámetro>)*);`

Entrada: scanf

- `scanf("<formato>"(, <parámetro>)*);`

Formato: string mezclado con identificadores de tipo:

`%d` – int

`%c` – char

`%f` – float

`%s` – string

En Python, no usamos formato (necesariamente) y usamos **print** e **input**



# Entrada / Salida

Dado que el formato no es obligatorio, se le da al texto, no a la función print:

- `print("{0}".format(, <parámetro>)*)`

Lo anterior implica que input **siempre** retorna un **string** → se debe convertir **explícitamente**

Funciones: `int()`, `float()`, `bool()`

# Tipos de datos especiales en C

- Punteros
- Strings

# Punteros

Una manera de hacer referencia a espacios de memoria

Se definen con un tipo de datos para saber cuanta memoria leer

Ejemplo:

```
int i; /*define una variable de tipo int*/  
int *p; /*define una var puntero a un int*/
```

# Los punteros son variables

Puedo asignarles un valor:

- Una dirección de memoria de otra variable usando el operador &
- Un nombre de arreglo
- Mediante aritmética de punteros
- Usando la función malloc()
- En el caso de variables de tipo char\*, string literales  
(`char* demo = "demo";`)

**NOTA:** En el caso de string literales, quedan sólo de lectura → NO SE PUEDEN MODIFICAR

# Strings

Cadenas de caracteres: arreglos o punteros a

Terminan con null ('\0')

Métodos (<string.h>)

`strcpy`

`strlen`

`strcmp`

`strcat`

...

# Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main() {
    char *s;
    char word[20];
    char *hi = "hello";
    char bye[] = "goodbye";
```

```
s = (char*)malloc(20*sizeof(char));
strcpy( s, "good morning");
printf( "%lu\n", strlen( s ) );
strcpy( word, "how are you?" );
printf( "%lu\n", strlen( word ) );
printf( "%c\n", s[4] );
s++;
printf( "%c\n", *s );
printf( "%lu\n", strlen( s ) );
}
```

# En Python

A diferencia de otros lenguajes como C, en Python el texto es un tipo más: se compara con `==`, se concatena con `+`

`len(string)` → largo del texto

¡SIN EMBARGO! String es una clase (¡orientado a objetos!), por lo que tiene métodos propios que veremos pronto

# char\* vs char[]

## Tarea con bonificación

```
#include <stdlib.h>
#include <stdio.h>

char* arreglo() {
    char texto[10] = "hola";
    return texto;
}
```

```
char* puntero() {
    char* texto = "hola";
    return texto;
}

int main(void) {
    char* p = puntero();
    printf("%s\n", p);
    char* a = arreglo();
    printf("%s\n", a);
}
```



# Iterable en Python

Python tiene un par de tipos especiales de datos: iterables y generadores. Ambos representan una secuencia de elementos

Ejemplo:

`range(0, 100, 2)` ← obtiene los números del 0 al 99 de 2 en 2

# Instrucciones en C

1. Expresión: sentencia simple que termina en un ;
2. Bloques de código: conjunto de 0 o más sentencias encerradas entre { }

# Instrucciones en C

**if:**

```
if (<condición>) instrucción  
else instrucción
```

<condición> debe  
evaluarse a un valor  
numérico

**switch:**

```
switch (<expresión>) {  
    case <constante>: instrucción  
    break;  
    default: instrucción  
}
```

break; hace que no se  
ejecuten todos los  
casos en forma  
secuencial

# Instrucciones

**while:**

```
while (<condición>) instrucción
```

**do-while:**

```
do instrucción while (<condición>);
```

**for:**

```
for (<inicialización>; <condición>;  
<asignación>) instrucción
```

while evalúa la condición antes de ejecutar las instrucciones. do/while primero ejecuta las instrucciones y luego evalúa la condición

Usen for cuando exista una variable de control clara.

# Instrucciones

## **break;**

Termina de iterar en un ciclo, o de evaluar en un switch/case

## **continue;**

Salta a la siguiente iteración en un ciclo

## **return (<expresión>);**

Termina la ejecución de un método, pudiendo retornar un valor en el caso de una función

# Ejemplo

```
#include <stdio.h>
int selected = 8;
int main(void) {
    int guess = -1;
    while( guess != selected ) {
        printf("Adivina mi número (entre 1 y 10): ");
        scanf("%d", &guess);
    }
    printf("¡Adivinaste!\n");
}
```

# Ejemplo

```
#include <stdio.h>
int selected = 8;
int main(void) {
    int guess;
    while(true) {
        printf("Adivina mi número (entre 1 y 10): ");
        scanf("%d", &guess);
        if ( guess == selected ) break;
    }
    printf("¡Adivinaste!\n");
}
```

# Instrucciones en Python

1. Expresión: sentencia simple, típicamente en una línea
2. Bloques de código: conjunto de 0 o más sentencias con la misma indentación



# Métodos en C

Declaración: `<tipo> nombre(<parámetros>);`

```
void push(char c);
```

Definición: declaración + cuerpo del método

```
void push(char c) {  
    store[top++] = c;  
}
```

Un método debe poseer:

- Tipo de datos de retorno (void si es procedimiento)
- Nombre
- Parámetros
- Cuerpo en un bloque

# Parámetros

Datos que se entregan a los métodos:

- Copia (paso por valor) → todo en C se pasa por valor
- Original (paso por referencia) → punteros

# Métodos en Python

Definición: declaración + cuerpo del método

```
def nombre(<parámetros>):
```

```
    store[top] = c
```

```
    top = top + 1
```

Un método debe:

- Comenzar por el keyword **def**
- Poseer Nombre
- Declarar Parámetros
- Tener un cuerpo en un bloque

# Parámetros

Datos que se entregan a los métodos:

- En Python los parámetros se pasan acorde a su definición
  - Si es inmutable: se crea una copia al modificarlo
  - Si no, se pasa por referencia (la variable original)

# ¿Immutable?

Variables a las que NO se les permite cambiar su valor después de su creación, a menos que se re-asignen

	Description	Immutable?
<b>bool</b>	Boolean value	✓
<b>int</b>	integer (arbitrary magnitude)	✓
<b>float</b>	floating-point number	✓
<b>list</b>	mutable sequence of objects	
<b>tuple</b>	immutable sequence of objects	✓
<b>str</b>	character string	✓
<b>set</b>	unordered set of distinct objects	
<b>frozenset</b>	immutable form of set class	✓
<b>dict</b>	associative mapping (aka dictionary)	

# Alcance de variables

- Variables definidas en un bloque `{ ... }` son visibles sólo dentro de ese bloque
- variables locales
- Parámetros de un método son locales al bloque de cuerpo de ese método
  - Todo lo declarado fuera de métodos u otros bloques es accesible desde todos lados

# Ejemplo

```
#include <stdio.h>

void no_cambia( int d );
void cambia( int* d );
void tampoco(int* d);
void no_cambia( int d ) {
    d = 19;
}
void tampoco(int *d) {
    int i = *d;
    i = 20;
}
void cambia(int* d) {
    *d = 21;
}
```

```
int main( void ) {
    int numero = 17;
    printf("%d\n", numero);
    no_cambia(numero);
    printf("%d\n", numero);
    tampoco(&numero);
    printf("%d\n", numero);
    cambia(&numero);
    printf("%d\n", numero);
}
```

# Creando módulos

Como vimos anteriormente, el código en C puede ser separado en encabezado y cuerpo

- El encabezado puede escribirse en otro archivo, con extensión .h (header file)
- El cuerpo incluye las declaraciones al hacer un include de ese encabezado
- Esto permite compartir código



# Ejemplo en C

push.c

```
#define MAX 100
#include <stdio.h>

void push( char c );

int top;
char store[MAX];

void push( char c ) {
    if (top < MAX)
        Store[top++] = c;
}
```

# Ejemplo

## push.h

```
#define MAX 100
#include <stdio.h>
void push( char c );
int top;
char store[MAX];
```

## push.c

```
#include "push.h"

void push( char c ) {
    if (top < MAX)
        store[top++] = c;
}
```

# Ejemplo

main.c

```
#include "push.h"
```

```
int main( void ) {  
    push('a');  
}
```

# Ejemplo en Python

push.py

```
import numpy
```

```
MAX = 100
```

```
top = 0
```

```
store = numpy.zeros(MAX)
```

```
def push( c ):
```

```
    global top
```

```
    if (top < MAX)
```

```
        store[top] = c
```

```
        top = top + 1
```

# Ejemplo

main.py

```
import push
```

```
if __name__ == "__main__":  
    push.push('a')
```

# ¿Qué vimos?

- Ejemplos en C y Python

# Tarea 1

Strcmp: strcmp es una función en C que recibe como parámetros 2 strings y compara que sean iguales. La idea de esta tarea es re-implementar strcmp en C y Python de forma que retorne que dos strings son iguales a pesar de tener case distinto. Por ejemplo HOla es igual a hola, HOla, HoLa, etc.

# ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	



# Declaración en C

```
int strcmpic(char *str1, char *str2);
```

Parámetros:

- str1 y str2 son los strings a comparar

Retorna:

- 1 si son iguales
- 0 si no lo son

# Tarea 1

Debes crear esta función en un módulo my\_strings, el cuál debe ser usado acá:

```
#include "my_strings.h"
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    char* str1 = "hola";
    char* str2 = (char*) malloc( 20 * sizeof(char));
    printf("Escribe hola:");
    scanf("%s", str2);
    if (strcmp(str1, str2) == 0) printf("no escribiste hola :( \n");
}
```

# Tarea 1

Pasos:

1. Crea un archivo `main.c` con el código de la slide anterior
2. Crea un archivo `my_strings.h` donde pegues la declaración de la función `strcmpic`
3. Crea un archivo `my_strings.c` donde defines la función `strcmpic` (declaración + cuerpo)

# Tarea 1

Para implementar strcmpic, debes:

- a. crear una copia de str1 y str2
- b. Mientras lo apuntado por str1 no sea `\0` y lo apuntado por str2 no sea `\0`
- c. Si lo apuntado por str1 está en mayúsculas, pasarlo a minúsculas. Lo mismo para str2
- d. Comparar lo apuntado por str1 con lo apuntado por str2. Si no es igual, retornan 0
- e. Si son iguales, entonces avanzar un caracter

# ¿Preguntas?