Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

Clase 15: Ejercicios Midterm

Rodrigo Toro Icarte (rntoro@uc.cl)

IIC1103 Introducción a la Programación - Sección 5

11 de Mayo, 2015

Control de flujo

"Cree una función que reciba un entero y retorne True si es primo."

Control de flujo

"Cree una función que reciba un entero y retorne True si es primo."

```
1 def es_primo(n):
2   if(n == 1): return False
3   for i in range(2,n):
4     if(n%i == 0):
5       return False
6   return True
7
8 n = int(input("Ingrese número: "))
9 print(es_primo(n))
```

Control de flujo

Teo: Un número primo p es expresable como suma de dos cuadrados si y sólo si $p = 2$ o $p - 1$ es divisible por 4.

Ej: $29 = 2^2 + 5^2 = 4 \cdot 7 + 1$ $\quad$ $41 = 4^2 + 5^2 = 4 \cdot 10 + 1$

Escribe un programa que pida al usuario un entero n e imprima los primeros n números primos que satisfacen el teorema, indicando cuál es su descomposición. Ejemplo:

```
> Ingrese un número: 3
> 2 = 1^2 + 1^2
> 5 = 1^2 + 2^2
> 13 = 2^2 + 3^2
```

Control de flujo

```
n = int(input("Ingrese número: "))
i = 2; contador = 0
while(contador < n):
  # verifico si es primo
  es_primo = True
  for j in range(2,i-1):
    if(i % j == 0):
      es_primo = False
  # Veo si cumple reglas del teorema
  teo = ((i == 2) or (i-1)%4 == 0)
  # si es primo y cumple teo, veo su descomposición
  if(es_primo and teo):
    for a in range(1,i):
      for b in range(a,i):
        if(a**2 + b**2 == i):
          print(i,"=",a,"^2 +",b,"^2")
    contador += 1
  # paso al siguiente número
  i += 1
```

Ejercicio funciones

"Programe un algoritmo capaz de multiplicar matrices."

Ejercicio funciones

"Programe un algoritmo capaz de multiplicar matrices."

Para codificar una matriz de $N \times M$ usamos una tupla:
(N,M,m), donde m es una lista con los elementos de la matriz.

Ejercicio funciones

"Programe un algoritmo capaz de multiplicar matrices."

Para codificar una matriz de $N \times M$ usamos una tupla: (N,M,m), donde m es una lista con los elementos de la matriz.

Ejemplo: La matriz:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

... queda codificada como: $\mathbf{m = (2,3,[1,2,3,4,5,6])}$.

Ejercicio funciones

Recuerde: El resultado de multiplicar dos matrices A (de $N \times M$) y B (de $M \times P$) es una matriz C de $N \times P$ tal que:

$$C_{i,j} = A_{i,:} \cdot B_{:,j}$$

Ejercicio funciones

Recuerde: El resultado de multiplicar dos matrices A (de $N \times M$) y B (de $M \times P$) es una matriz C de $N \times P$ tal que:

$$C_{i,j} = A_{i,:} \cdot B_{:,j}$$

Ejemplo:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2\times 3} \cdot \begin{bmatrix} 8 & 0 \\ 5 & -1 \\ 2 & 3 \end{bmatrix}_{3\times 2} = \begin{bmatrix} 24 & 7 \\ 69 & 13 \end{bmatrix}_{2\times 2}$$

Notar que $C_{2,2} = (4\ 5\ 6) \cdot (0\ \ -1\ 3) = 4 \cdot 0 + 5 \cdot -1 + 6 \cdot 3 = 13$.

Ejercicio funciones

Para ello implemente las siguientes funciones:

- **producto_punto(a,b)** que recibe 2 tuplas y retorna el producto punto entre ellas.
- **obtener_fila(m,i)** que retorna la fila **i** de la matriz **m**.
- **obtener_col(m,j)** que retorna la columna **j** de la matriz **m**.
- **multiplicar(m1,m2)** que retorna la multiplicación de las matrices **m1** y **m2**.

Ejercicio funciones

Ejemplo:

```
36 m1 = (2,3,[1,2,3,4,5,6])
37 m2 = (3,2,[8,0,5,-1,2,3])
38 print(multiplicar(m1,m2))
39 # >>> (2, 2, [24, 7, 69, 13])
```

Ejercicio funciones

producto_punto(a,b) que recibe 2 tuplas y retorna el producto punto entre ellas.

Ejercicio funciones

producto_punto(a,b) que recibe 2 tuplas y retorna el producto punto entre ellas.

```
2 def producto_punto(a,b):
3   p = 0
4   for i in range(len(a)):
5     p += a[i]*b[i]
6   return p
```

Ejercicio funciones

obtener_fila(m,i) que retorna la fila **i** de la matriz **m**.

Ejercicio funciones

obtener_fila(m,i) que retorna la fila **i** de la matriz **m**.

```
9  def obtener_fila(m,i):
10   N = m[0]
11   M = m[1]
12   return m[2][M*(i-1):M*i]
```

Ejercicio funciones

obtener_fila(m,i) que retorna la fila **i** de la matriz **m**.

```
9  def obtener_fila(m,i):
10   N = m[0]
11   M = m[1]
12   return m[2][M*(i-1):M*i]
```

obtener_col(m,j) que retorna la columna **j** de la matriz **m**.

Ejercicio funciones

obtener_fila(m,i) que retorna la fila **i** de la matriz **m**.

```
9  def obtener_fila(m,i):
10   N = m[0]
11   M = m[1]
12   return m[2][M*(i-1):M*i]
```

obtener_col(m,j) que retorna la columna **j** de la matriz **m**.

```
15 def obtener_col(m,j):
16   N = m[0]
17   M = m[1]
18   return m[2][j-1::M]
```

Ejercicio funciones

multiplicar(m1,m2) que retorna la multiplicación de las matrices **m1** y **m2**.

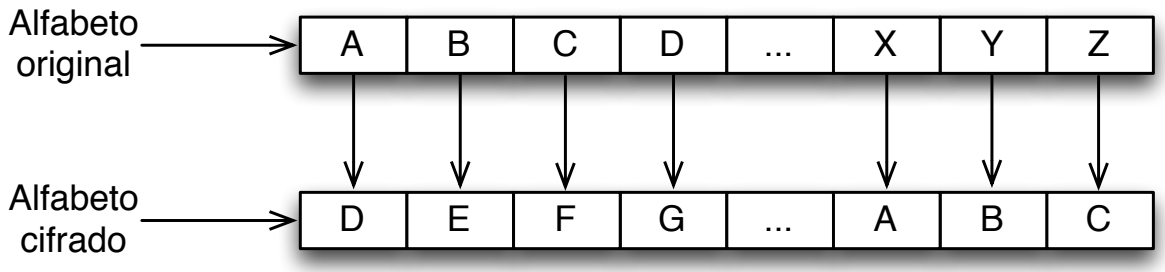
Ejercicio funciones

multiplicar(m1,m2) que retorna la multiplicación de las matrices **m1** y **m2**.

```
21 def multiplicar(m1,m2):
22   # Obtengo dimensiones nueva matriz y la defino
23   N = m1[0]; M = m2[1]; m = []
24
25   # Para formar nueva matriz multiplico filas de m1
       por columnas de m2
26   for i in range(1,N+1):      # recorro filas
27     for j in range(1,M+1):    # recorro columnas
28       fila_i = obtener_fila(m1,i)
29       col_j = obtener_col(m2,j)
30       res = producto_punto(fila_i,col_j)
31       m.append(res)
32
33   return (N,M,m)
```

Ejercicio strings

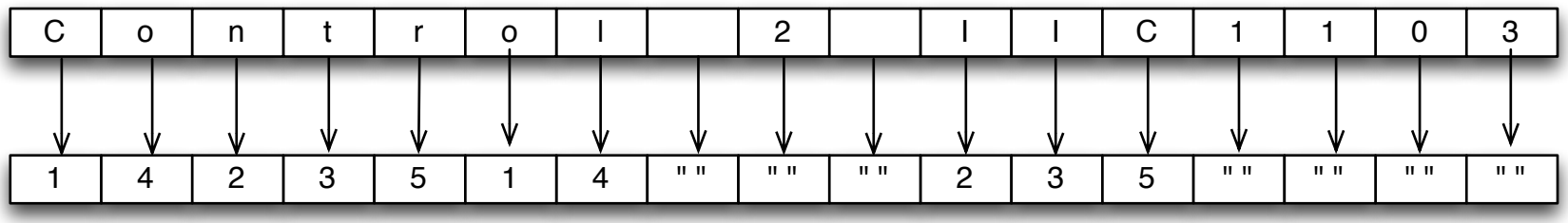
El encriptador `ROT-n` consiste en tomar un string y cambiar cada caracter por el que se encuentre `n` posiciones delante de él.



Programa una variante de `ROT-n` que reciba una clave numérica que indique cuánto se desplaza cada caracter alfabético.

Ejercicio strings

Ejemplo: Si la clave es `14235` el primer caracter se desplaza 1, el segundo 4, ..., y luego del 5 se regresa al 1.



`Observaciones`:

- Hay que distinguir entre mayúsculas y minúsculas.
- Solo se consideran caracteres alfabéticos (sin eñes ni tildes).
- Caracteres no alfabéticos se dejan tal cual y no se avanza en la clave.

Ejercicio strings

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

Ejercicio strings

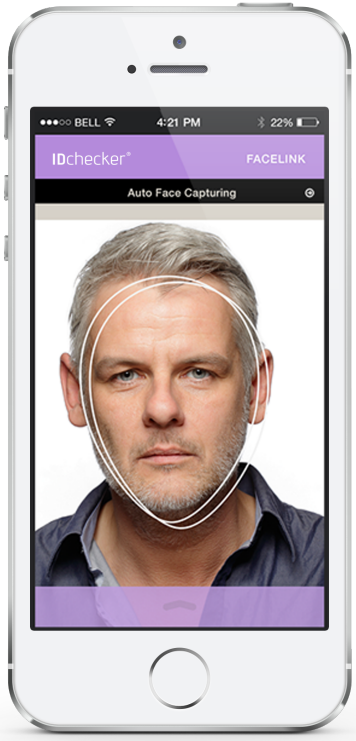
```
1  def mover(c,i,a,z):
2    id_c = ord(c) + i
3    if(id_c > ord(z)):
4      id_c -= ord(z) - ord(a) + 1
5    return chr(id_c)
6
7  def encriptar(mensaje, clave):
8    r = ""; clave = str(clave); clave_id = 0
9    for i in range(len(mensaje)):
10     c = mensaje[i]
11     c_i = int(clave[clave_id % len(clave)])
12     if(ord('a') <= ord(c) <= ord('z')):
13       c = mover(c,c_i,'a','z')
14       clave_id += 1
15     elif(ord('A') <= ord(c) <= ord('Z')):
16       c = mover(c,c_i,'A','Z')
17       clave_id += 1
18     r += c
19   return r
```

Ejercicio listas

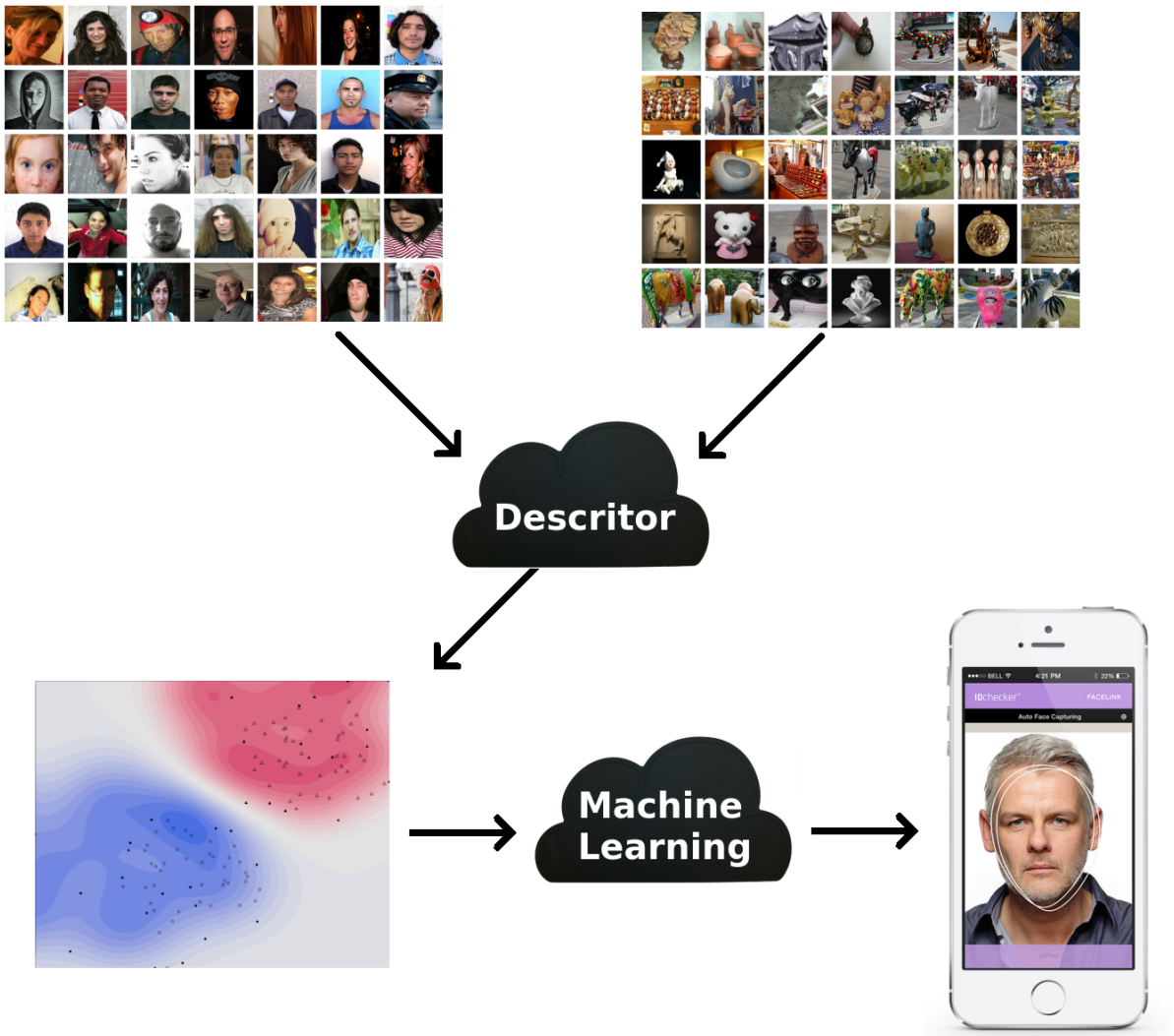
"Programe el algoritmo conocido como vecinos cercanos."

Ejercicio listas

"Programe el algoritmo conocido como vecinos cercanos."



Ejercicio listas

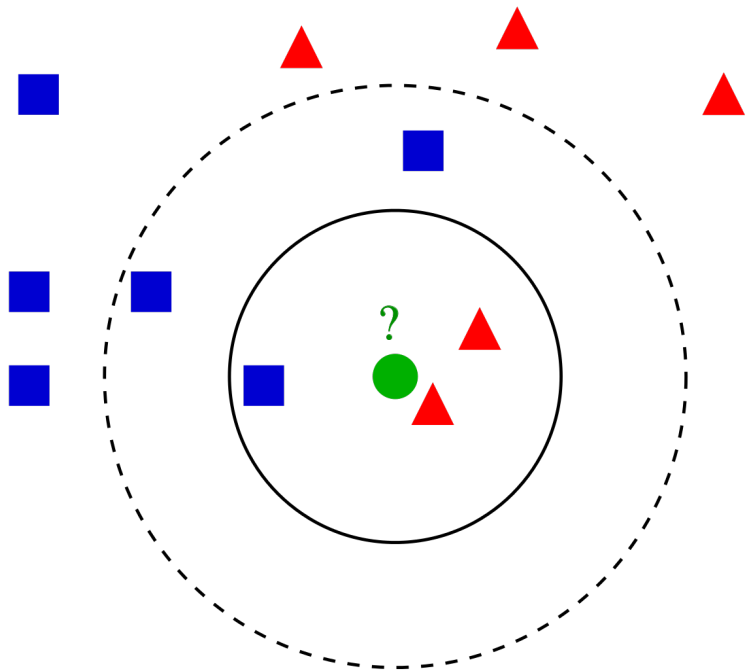


Ejercicio listas

Vecinos cercanos: Para decidir si un nuevo *punto* es una cara o no, encontraremos los k puntos más cercanos y le asignamos la clase de sus vecinos.

Ejercicio listas

Vecinos cercanos: Para decidir si un nuevo *punto* es una cara o no, encontraremos los k puntos más cercanos y le asignamos la clase de sus vecinos.



Ejercicio listas

Implemente una función que reciba una lista **L** de *puntos rotulados* (tuplas: ((x0,x1,x2,...),clase)), un valor para **k**, un punto **p** (tupla (x0,x1,x2,...)) y retorne True ssi **p** es una cara.

Ejercicio listas

Implemente una función que reciba una lista **L** de *puntos rotulados* (tuplas: $((x_0,x_1,x_2,...),\text{clase})$), un valor para **k**, un punto **p** (tupla $(x_0,x_1,x_2,...)$) y retorne True ssi **p** es una cara.

```
41 caras = [((1,3,2),'+'),((2,2,2),'+'),
42          ((4,0,3),'+'),((5,5,0),'+'),
43          ((3,2,1),'+'),((4,3,6),'+')]
44
45 no_caras = [((30,10,21),'-'),((12,14,10),'-'),
46              ((21,0,33),'-'),((42,5,66),'-'),
47              ((33,21,7),'-'),((6,12,15),'-')]
48
49 L = caras + no_caras
50
51 nuevo = (4,4,4); k = 5
52 print(es_cara(nuevo,L,k))
```

Ejercicio listas

¿Cómo obtengo los **k** vecinos más cercanos a **p** en **L**?

Ejercicio listas

```
19 def es_cara(punto,L,k):
20   # obtengo los k vecinos más cercanos
21   vecinos = []
22   for p in L:
23     d = distancia(p[0],punto)
24     if(len(vecinos) < k):
25       vecinos.append(p)
26     else:
27       val_max,vec_max = get_max(vecinos, punto)
28       if(d < val_max):
29         vecinos.remove(vec_max)
30         vecinos.append(p)
31
32   # Veo la clase
33   num_caras = 0
34   for p in vecinos:
35     if(p[1] == '+'): num_caras += 1
36
37   return num_caras >= len(vecinos)/2
```

Ejercicio listas

Función que calcula distancia entre 2 puntos.

Ejercicio listas

Función que calcula distancia entre 2 puntos.

```
2 def distancia(p1,p2):
3   ret = 0
4   for i in range(len(p1)):
5     ret += (p1[i]-p2[i])**2
6   return ret**0.5
```

Ejercicio listas

Función que calcula distancia entre 2 puntos.

```
2 def distancia(p1,p2):
3   ret = 0
4   for i in range(len(p1)):
5     ret += (p1[i]-p2[i])**2
6   return ret**0.5
```

Función que retorna el vecino a distancia máxima del punto.

Ejercicio listas

Función que calcula distancia entre 2 puntos.

```
2  def distancia(p1,p2):
3    ret = 0
4    for i in range(len(p1)):
5      ret += (p1[i]-p2[i])**2
6    return ret**0.5
```

Función que retorna el vecino a distancia máxima del punto.

```
9  def get_max(vecinos, punto):
10   val_max = -1
11   for p in vecinos:
12     d = distancia(p[0],punto)
13     if(val_max < d):
14       val_max = d
15       vec_max = p
16   return val_max,vec_max
```

Cierre

Miércoles Clase Práctica... ¡No falte!