ISE 101 – Introduction to Information Systems

- Lecture 3 Objectives:
  - While loops
  - Strings

While Loops

- Write a Python script that computes the sum of squares from 1 to 5.

```
sum = 0;

sum = sum + 1**2;
sum = sum + 2**2;
sum = sum + 3**2;
sum = sum + 4**2;
sum = sum + 5**2;

print('Sum of squares: ' + str(sum))
```

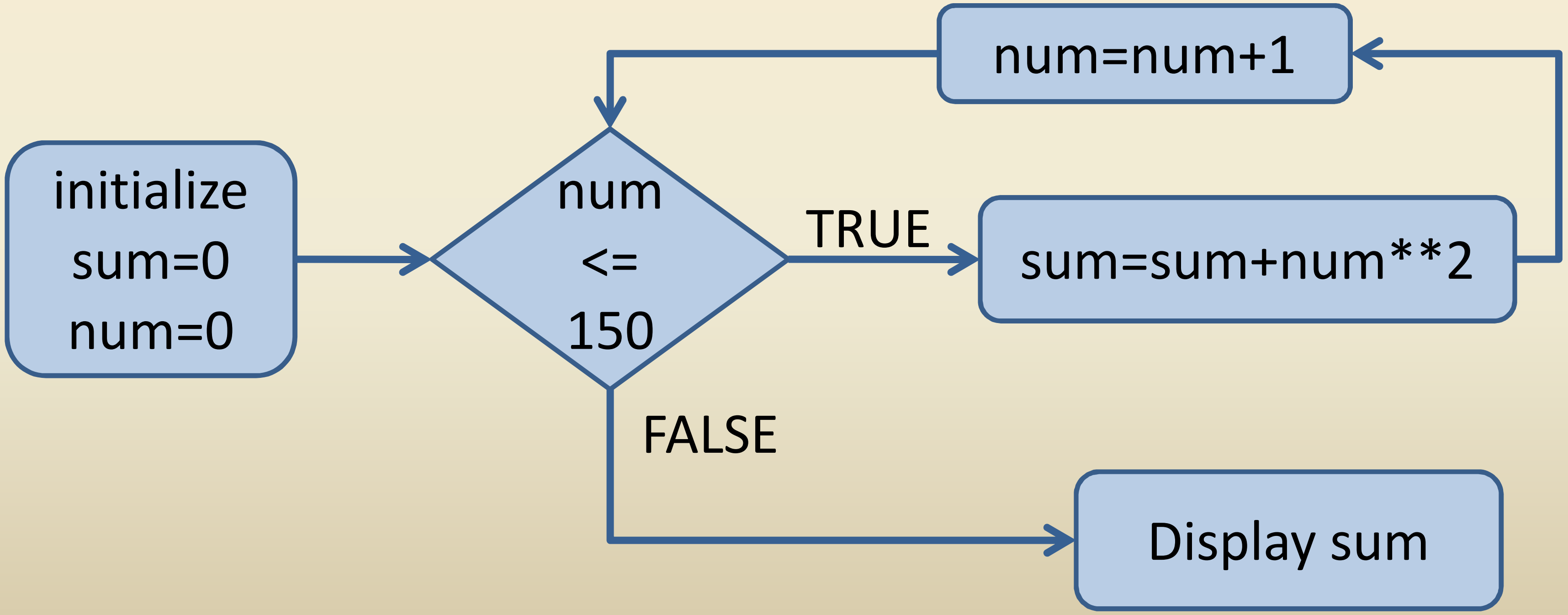
While Loops

- Write a Python script that computes the sum of squares from 1 to 150.

```
sum = 0;

sum = sum + 1**2;
sum = sum + 2**2;
…
sum = sum + 149**2;
sum = sum + 150**2;

print('Sum of squares: ' + str(sum))
```

Loops

- Sometimes a portion of the code has to be repeated many times.
- In the previous example, the square and summation have to be repeated 150 times.



While Loops

- Write a Python script that computes the sum of squares from 1 to 150.

```
sum = 0
num = 0
while num<=150:
      sum = sum + num**2
      num = num + 1

print('sum of squares: ' + str(sum))
```

While Loop

Use colon after the logical expression

```
while (logical_expression):
        statement 1
        statement 2
        statement 3
```

Statements in the scope are executed if the logical expression is True

TABS are used to determine the scope of while loop

While Loop

```
while (logical_expression):
        statement 1
        statement 2
        statement 3
else:
        statement 4
        statement 5
        statement 6
```

$\left.\begin{array}{l}\text{statement 4}\\\text{statement 5}\\\text{statement 6}\end{array}\right\}$ Statements in this scope are executed if the logical expression is False

While Loop

- Statements within the scope of the while loop are executed if the logical statement produces TRUE
- If the logical statement generates FALSE, the statements within the scope are skipped.
- The result of the logical expression changes within the loop
- If the logical expression gives TRUE, it becomes an infinite loop
- Infinite loops never end

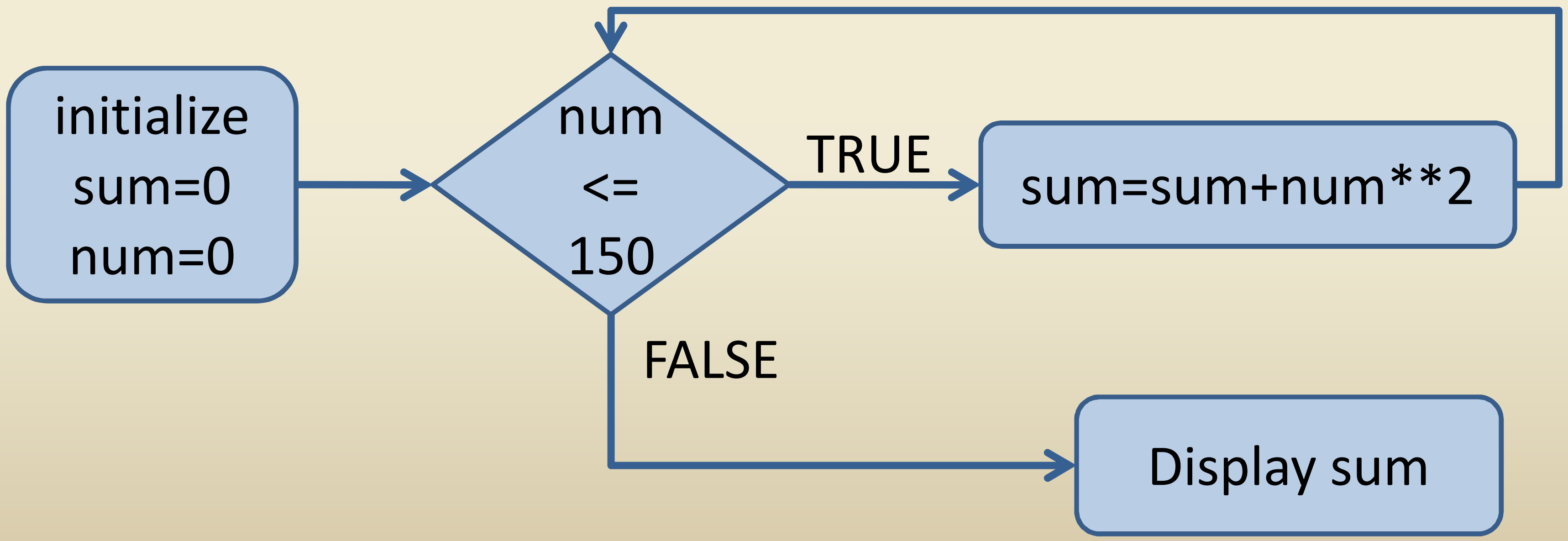
While Loop

```
while ( True ):
        statement 1
        statement 2
        statement 3
```

```
while (3 < 5):
        statement 1
        statement 2
        statement 3
```

Infinite Loops

- What is the problem with the following algorithm?
- Variable num is not incremented.
- Therefore, (num<=150) will always produce True.
- This is an infinite loop, that never ends.



Pre-test and Post-test Loops

Pre-test loop

- Pre-test loop controls a logical expression before the loop
- While loop is pre-test loop
- The logical statement after while is tested
- If it produces True, the statements within the scope are executed

Post-test loop

- Post-test loop controls a logical expression after the loop
- It executes the statements within the scope of the loop
- It checks a logical expression
- If it produces True, loop continue
- There is no post-test loop in Python
- In C, do/while loop is an example of post-test loop

Example

- Write a Python script that
  - generates a random number
  - asks the user to guess the random number until the guess is equal to the random number

```python
import random

random_number = random.randint(0,10)
user_guess=-1

while random_number!=user_guess:
    user_guess=int( input('Enter an integer : '))

print('Correct')
```

Example

- Write a Python script that computes the average of even integers between 1-100

```
sum = 0
num = 1
N=0

while num<=100:
    if num%2==0:
        sum = sum + num
        N = N+1

    num = num + 1

print('Average of even numbers between 1-100 : '
+ str(sum/N))
```

Example

- Write a Python script that gets floating point numbers from the user and adds them up until the sum is greater than 100

```
sum = 0

while sum<=100:
    num=float( input('Enter another number: '))
    sum = sum + num

print('Sum : ' + str(sum))
```

Example

- Write a Python script that finds and prints how many numbers there are between 1-1000 which are a multiple of 7.

```python
num=1
N_multiple=0

while num<=1000:
    if num%7==0:
        N_multiple = N_multiple + 1
    num = num + 1

print('There are ' + str(N_multiple) + ' multiples
of 7 between 1-1000')
```

Break

- Sometimes, the loop has to be stopped
- “break” is used within the loop to stop the loop
- Loop is terminated
- Once the loop is stopped using break, statements within the scope of the loop (that are after break) are skipped.

Example

```
denom=5

while (denom>=-2):
   if denom!=0:
      print('Result is :' + str(10/denom))
   else:
      print('Stopping the loop')
      break

   denom = denom - 1
   print('Still in the loop')

print('Loop is ended')
```

Output

```
Result is :2.0
Still in the loop
Result is :2.5
Still in the loop
Result is :3.3333333333333335
Still in the loop
Result is :5.0
Still in the loop
Result is :10.0
Still in the loop
Stopping the loop
Loop is ended
```

Continue

- In the previous example, results of 10/-1 and 10/-2 are not printed
- Because of break, the while loop is terminated when denom is 0
- Logical expression (denom>=-2) is still True
- To STOP the **current** loop iteration **WITHOUT** termination the loop “continue” is used
- Statements that are after the “continue” are skipped
- The logical statement is re-evaluated
- If it is True, loop continues
- If it is False, loop terminates

Example

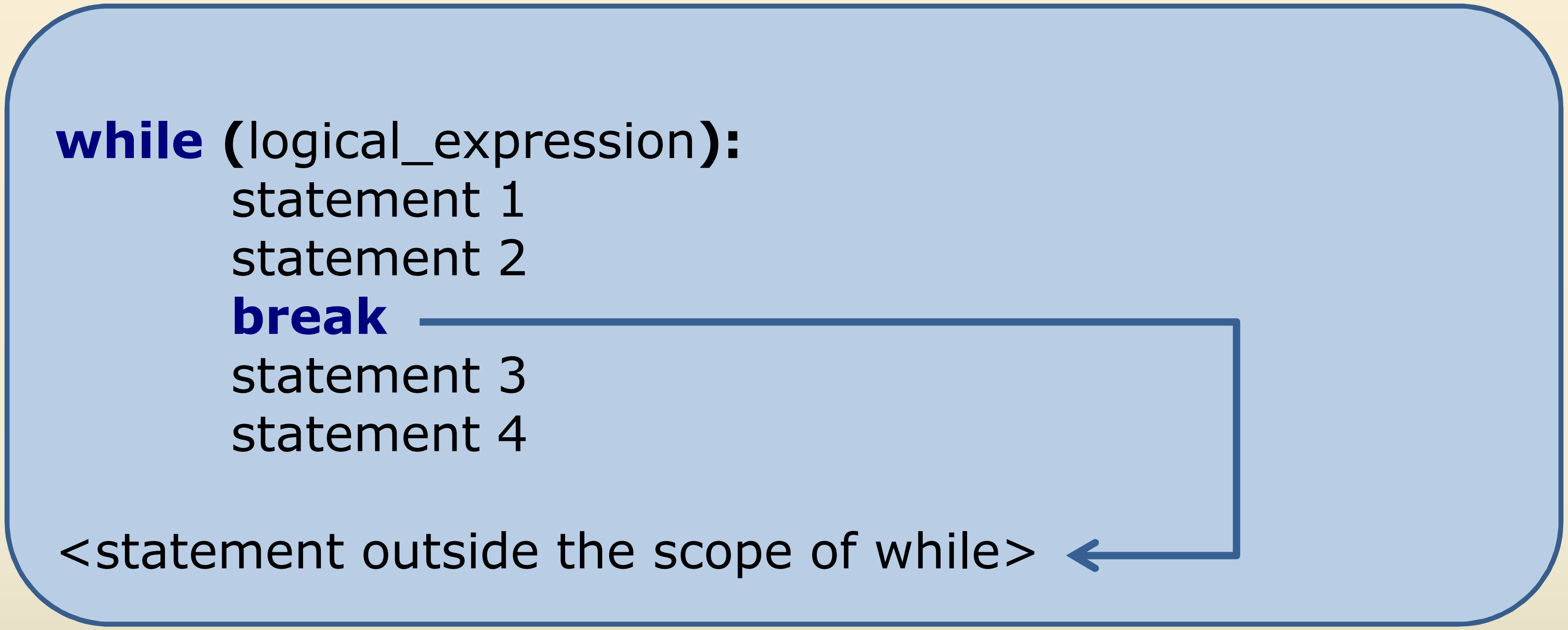
```
denom=5

while (denom>=-2):
   if denom!=0:
      print('Result is :' + str(10/denom))
   else:
      print('Stopping the loop')
      denom = denom - 1
      continue

   denom = denom - 1
   print('Still in the loop')

print('Loop is ended')
```

Output

```
Result is :2.0
Still in the loop
Result is :2.5
Still in the loop
Result is :3.3333333333333335
Still in the loop
Result is :5.0
Still in the loop
Result is :10.0
Still in the loop
Stopping the loop
Result is :-10.0
Still in the loop
Result is :-5.0
Still in the loop
Loop is ended
```

Difference of break and continue

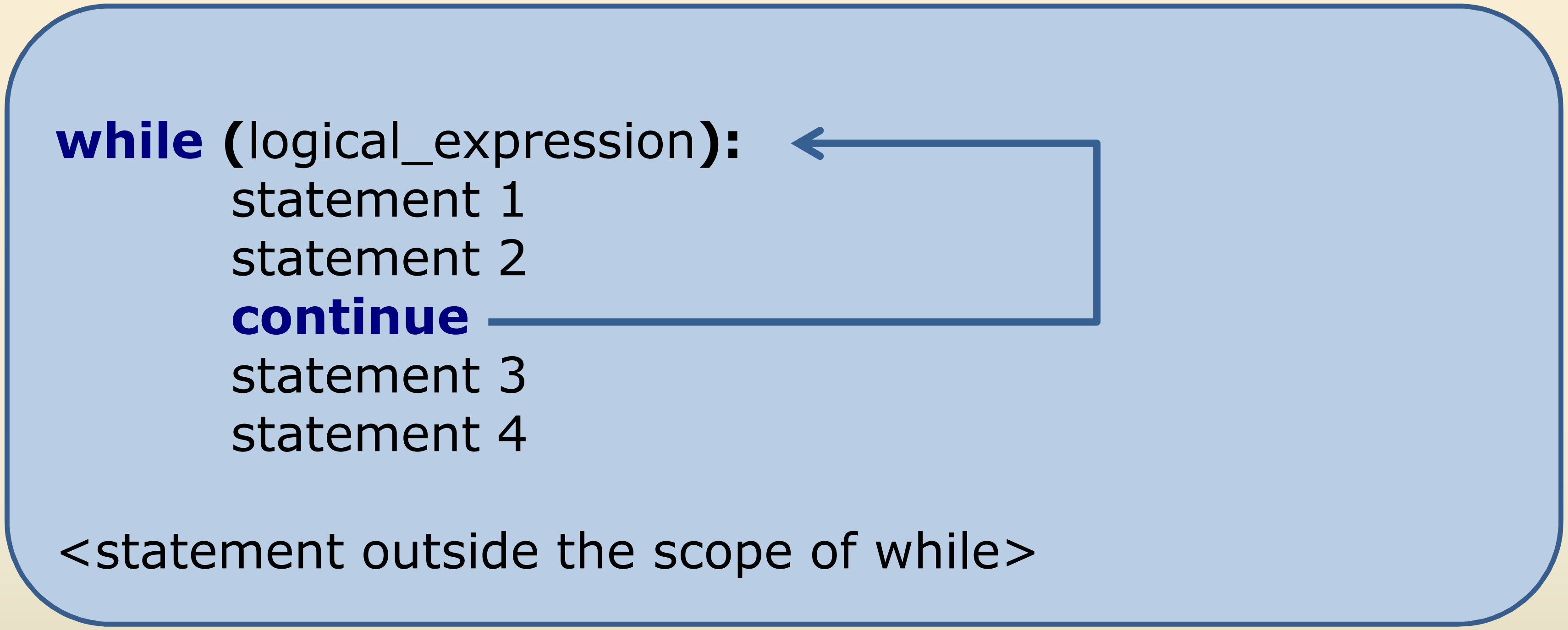
- break



```
while (logical_expression):
        statement 1
        statement 2
        break
        statement 3
        statement 4

<statement outside the scope of while>
```

Difference of break and continue

- continue



```
while (logical_expression):
        statement 1
        statement 2
        continue
        statement 3
        statement 4

<statement outside the scope of while>
```

STRINGS

Python Variable Types

- Until now, scalar data types are used
- Scalar ← variable stores a single value
  - X = 3 ← scalar
  - X = [3, 5] ←vector (in programming array and list are used instead of vector)
- Numeric data types
  - Integers
  - Floating point numbers (floats)
  - Complex numbers
- Boolean

Strings

- Strings are arrays of characters
- X='TEST STRING'

T	E	S	T		S	T	R	I	N	G

- Elements of an array can be reached using indexing

0	1	2	3	4	5	6	7	8	9	10
T	E	S	T		S	T	R	I	N	G

- X[index] : Character at the location specified by index
- index has to be integer!
  - X[0]='T'
  - X[7]=? 'R'

Strings

- Indices start with 0
- Index 0 corresponds to the first letter of the string
- T='New String'
- T[0] = ? 'N'
- T[15] = ? Error !!! Index out of range
- Be carefull not to use an index that is out of range
- len(variable_name) shows the length of the string that is in the variable
- So index should start from 0 and go upto len(variable_name)-1

Strings

```
>>> T='New String'
>>> len(T)
10
>>> T[0]
'N'
>>> T[len(T)-1]
'g'
>>> T[len(T)]
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.IndexError: string index out of range
```

Strings

- Contrary to other programming languages, negative indices can be used in Python
- Negative indices start from the back of the string

-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
T	E	S	T		S	T	R	I	N	G

- X[-1] = ‘G’
- X [-4] = ? ‘R’
- Negative indices start from -1 go down to -len(variable)
- Smaller indices than –len(variable) will be out of range and cause error

Strings

```
>>> T='New String'
>>> len(T)
10
>>> T[-1]
'g'
>>> T[-10]
'N'
>>> T[-11]
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.IndexError: string index out of range
```

String Slicing

- It is possible to index a substring
- This is called ‘string slicing’
- Instead of a single integer as an index, two integers separated by a column is used:

index1 : index2

Index of the first letter in the substring

Index of the last letter + 1 in the substring

String Slicing

- A slice 4:8 means indices: 4-5-6-7
- The second integer index (8 in the sample above) is not included
- It is possible to use negative indices for slicing

```
>>> T='This is a long test string'
>>> T[0:5]
'This '
>>> T[3:7]
's is'
>>> T[-5:-1]
'trin'
```

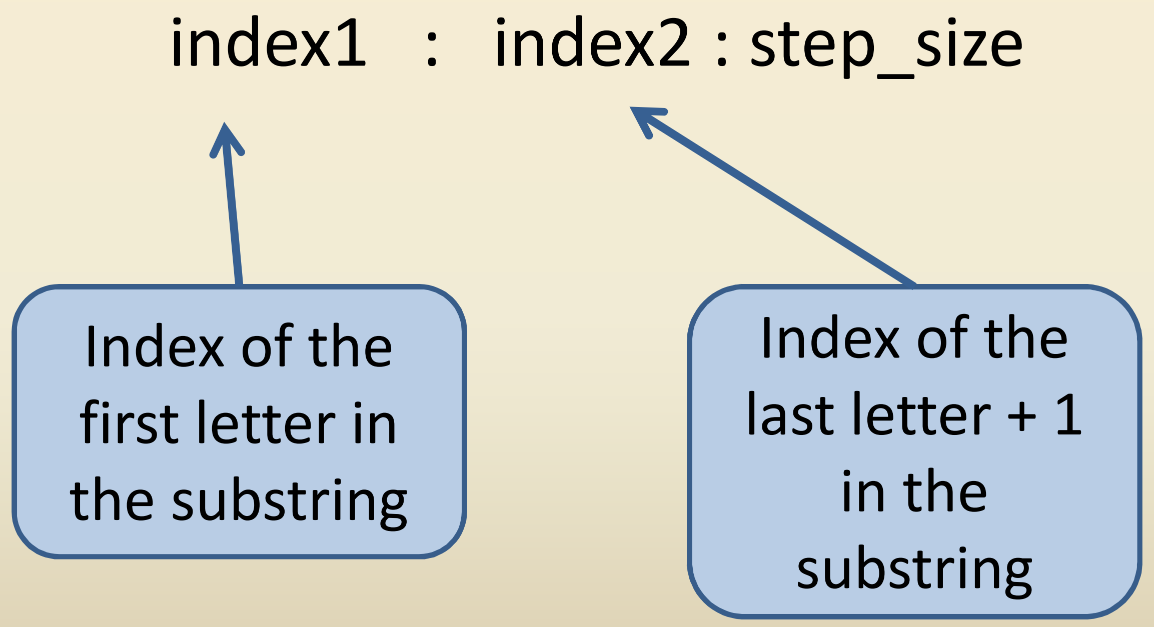
String Slices

- The second index should be smaller than the first
- Otherwise an empty string will be returned

```
>>> T='This is a long test string'
>>> T[5:4]
''
>>> T[-2:3]
''
```

String Slices

- Indices are not have to be sequential
- A third integer can be used to adjust the step size


index1 : index2 : step_size
Index of the first letter in the substring
Index of the last letter + 1 in the substring

String Slices

- For example

  3:8:2   (step size is 2)

  means indices

  3, 5, 7
- If no step size is given, it is assumed to be 1
- Step size can be negative
- For example

  -1:-7:-1

  means

  -1, -2, -3, -4, -5, -6

String Slices

```
>>> T='This is a long test string'
>>> T[2:9:3]
'iia'
>>> T[1:7]
'his is'
>>> T[11:7:-1]
'ol a'
>>> T[-1:-len(T)-1:-1]
'gnirts tset gnol a si sihT'
```

T[-1:-len(T)-1:-1] inverts the string

String Slices

- If the first index is skipped (empty) the first possible index will be used

```
>>> T='This is a long test string‘
>>> T[:5]
'This '
>>> T[:-1]
'This is a long test strin'
>>> T[:-10:-1]
'gnirts ts'
```

String Slices

- If the last index is skipped (empty) the last possible index + 1 (or -1) will be used

```
>>> T='This is a long test string‘
>>> T[2:]
'is is a long test string'
>>> T[2::-1]
'ihT'
>>> T[-5:]
'tring'
```

String Operators

- + and * operators are defined with strings
- However they have other meanings
  - ‘+’ means concatenation of strings
  - ‘*’ means repetition of a string
- Examples

```
>>> str1="Yellow"
>>> str2='car'
>>> print str1+str2
Yellowcar
```

```
>>> str2='car'
>>> 3*str2
'carcarcar'
```

Examples

```
>>> mystr='car'
>>> ('yellow'+mystr)*5
'yellowcaryellowcaryellowcaryellowcaryellowcar'
>>> 'yellow'+mystr*5
'yellowcarcarcarcarcar'
>>> 'yellow'*3+mystr*5
'yellowyellowyellowcarcarcarcarcar'
>>> ('yellow'*3+mystr)*2
'yellowyellowyellowcaryellowyellowyellowcar'
```

‘in’ Operator

- The operator ‘in’ can be used to check the existence of a character or a substring within a string
- ‘in’ expression return a Boolean value

```
>>> T='Test string'
>>> 'Y' in T
False
>>> 'R' in T
False
>>> 'es' in T
True
>>> 't s' in T
True
>>> 'esk' in T
False
```

String library

- There are many useful functions in string library

```
>>> import string
```

- String library has the following functions

Function	Meaning
capitalize(s)	copy of s, with first character is capitalized
capwords(s)	copy of s, with first character of each word is capitalized
center(s,width)	center s in a field of given width
count(s,sub)	count the number of occurances of substring sub in s
find(s,sub)	find the first position where sub occurs in s
join(list)	concatenate list of strings into one string
ljust(s,width)	left justify s in a field of given width
lower(s)	copy of s in all lowercase characters
lstrip(s)	copy of s with leading white space removed

String Library

Function	Meaning
replace(s,oldsub,newsub)	replace occurances of oldsub with newsub
rfind(s,sub)	like find but returns the righ-most position
rjust(s,width)	rigth justify s in a field of given width
rstrip(s)	sopy of s with trailing white space removed
split(s)	splits s into a list of substrings
upper(s)	copy of s with all characters converted to upper case letters

Examples

- You can use these functions in two different ways

```
>>> testString="today is rainy"
>>> string.upper(testString)
'TODAY IS RAINY'
>>> print(testString)
today is rainy
>>> testString.upper()
'TODAY IS RAINY'
>>> print(testString)
today is rainy
```

Examples

'

```
>>> string.split(testString)
['today', 'is', 'rainy']
>>> string.find(testString,'a')
3
>>> string.rfind(testString,'a')
10
>>> string.replace(testString,'a',"*")
'tod*y is r*iny‘
>>> string.center(testString,50)
'                today is rainy                ‘
```

Examples

- Some of these function can take arguments to refine their execution
- ‘find’ method takes a second argument the index where it should start searching

```
>>> str='Besiktas'
>>> str.find('s',3)
7
```

- Third argument is the index where find should stop searching

```
>>> str.find('s',3,5)
-1
```

Examples

- Second argument for split tells it which character /substring to use for splitting

```
>>> string.split(testString,'a')
['tod', 'y is r', 'iny']
>>> string.split(testString,'ay')
['tod', ' is rainy']
```

Immutable

- Strings are immutable (read-only) arrays
- Once a string is generated,
  - its characters can be accessed using indexes
  - However, a character within the string cannot be changed

```
>>> T='Test string'
>>> T[5]
's'
>>> T[5]='K'
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
builtins.TypeError: 'str' object does not support
   item assignment
```

String Comparison

- Two strings can be compared using
  - '==', '!='
  - '<', '>', '<=', '>='
- If two strings are the same '==' returns True, otherwise it returns False

```
>>> 'a' == 'b'
False
>>> 'a' == 'A'
False
'a'=='a'
True
' a' == 'a'
False
```

String Comparison

- If the strings are compared for equality, all characters (including white spaces) should be the same.
- ‘!=‘ is used to check if the strings are NOT the same.
- If strings are not equal, it returns True.
- If they are equal, it returns False.

```
>>> ‘a’ != ‘b’
True
>>> ‘a’ == ‘A’
True
‘a’==‘a’
False
‘ a’ == ‘a ’
True
```

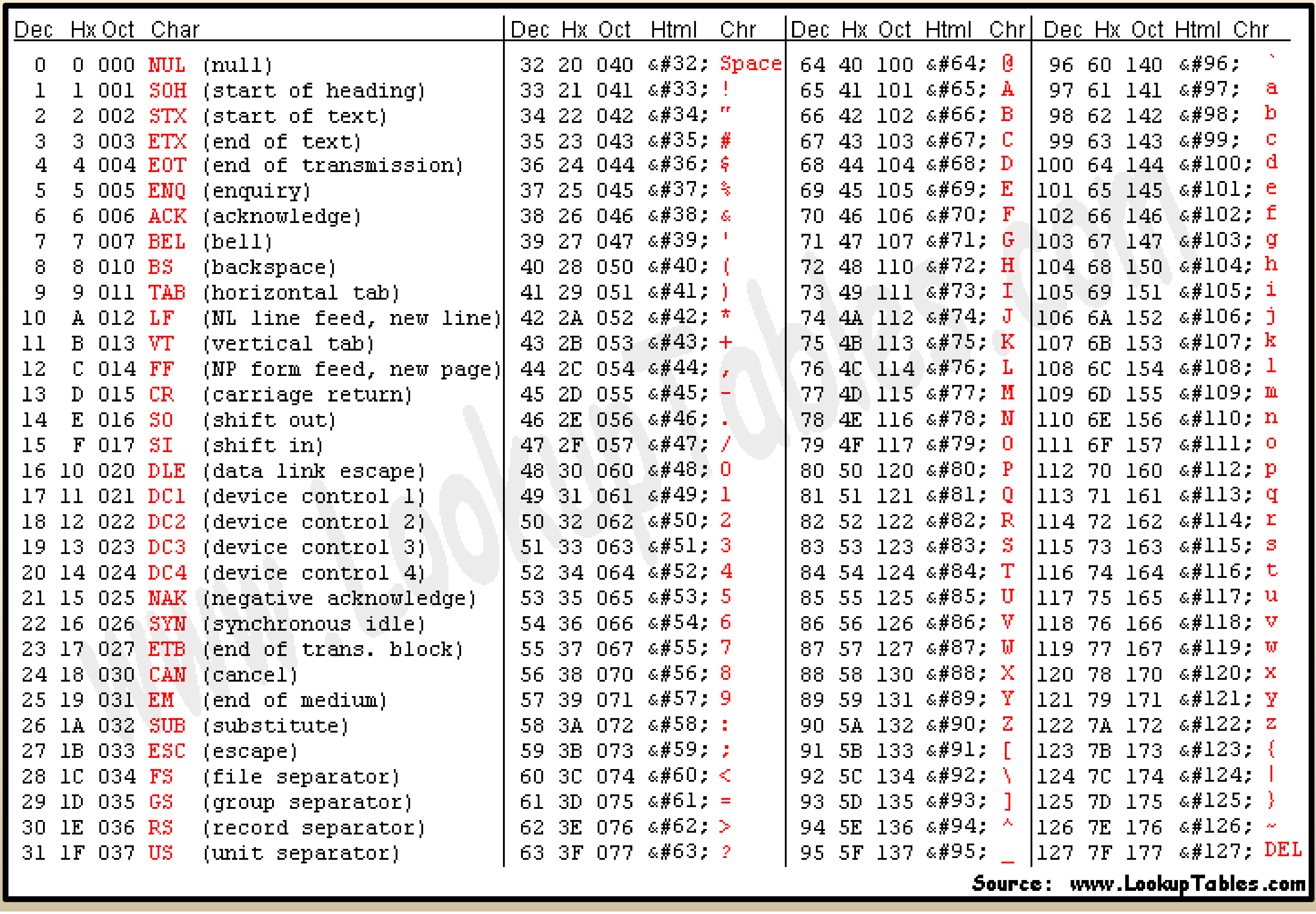
String Comparison

- How can we compare two strings for being greater or smaller?
- ‘a’ < ‘b’ ? True or False
- ASCII were used to compare strings in Python 2.x
- Python 3.x uses Unicode encoding for strings
- To compare two strings
  - Unicode for the first letter of both strings are compared. If one is bigger than the other, the string with the ‘bigger’ first letter is greater than the other. Remaining letters of the strings are skipped.

String Comparison

- If first letters are the same, second letters are compared. If one is bigger than the other, the string with the ‘bigger’ second letter is greater than the other. Remaining letters of the strings are skipped.
- If second letters are also the same, third letters are compared. If one is bigger than the other, the string with the ‘bigger’ third letter is greater than the other. Remaining letters of the strings are skipped.
- …

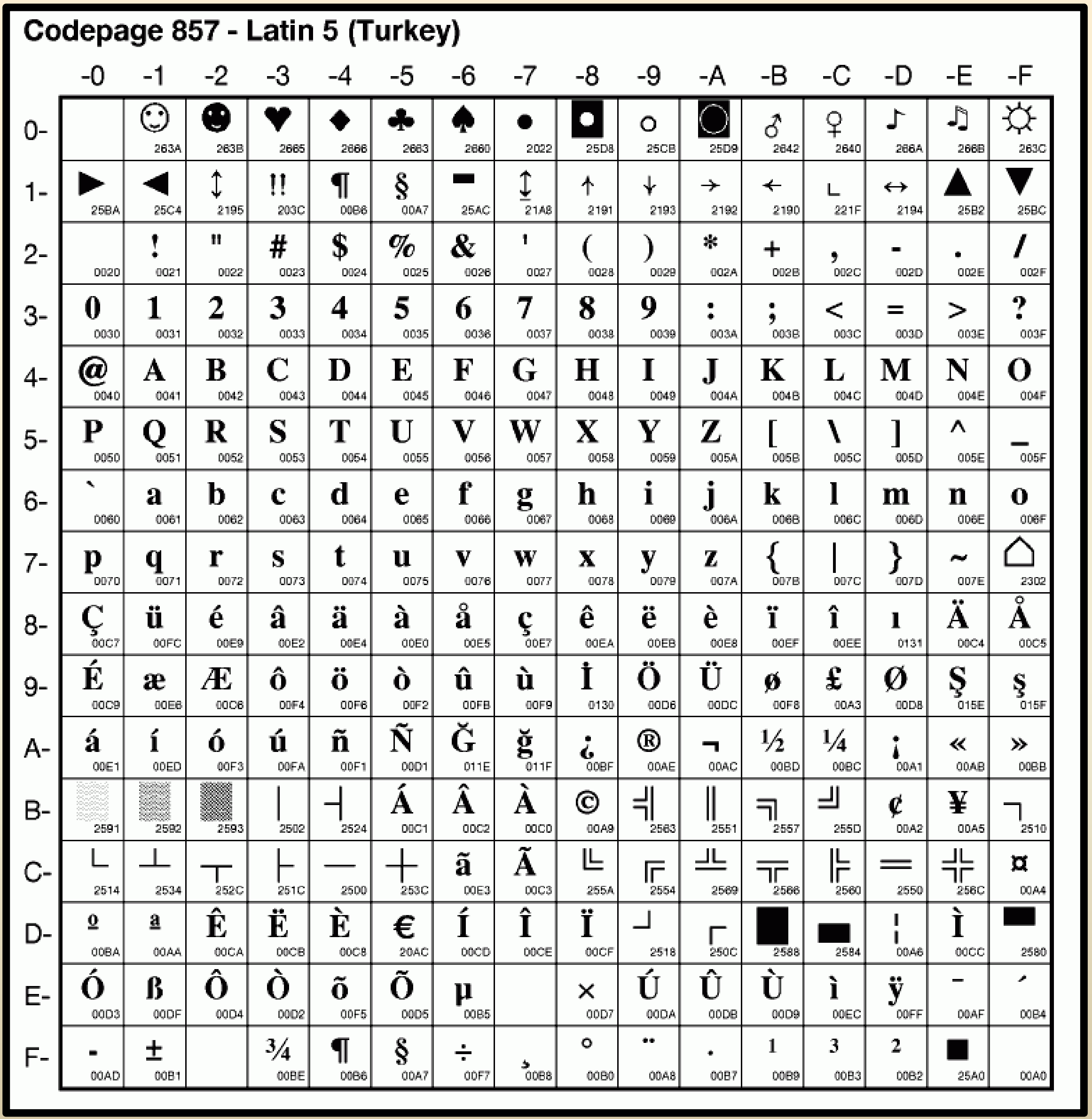
ASCII Table



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Unicode



Codepage 857 - Latin 5 (Turkey)

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-		☺ 263A	☻ 263B	♥ 2665	♦ 2666	♣ 2663	♠ 2660	• 2022	◘ 25D8	○ 25CB	◙ 25D9	♂ 2642	♀ 2640	♪ 266A	♫ 266B	☼ 263C
1-	► 25BA	◄ 25C4	↕ 2195	‼ 203C	¶ 00B6	§ 00A7	▬ 25AC	↨ 21A8	↑ 2191	↓ 2193	→ 2192	← 2190	∟ 221F	↔ 2194	▲ 25B2	▼ 25BC
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	⌂ 2302
8-	Ç 00C7	ü 00FC	é 00E9	â 00E2	ä 00E4	à 00E0	å 00E5	ç 00E7	ê 00EA	ë 00EB	è 00E8	ï 00EF	î 00EE	ı 0131	Ä 00C4	Å 00C5
9-	É 00C9	æ 00E6	Æ 00C6	ô 00F4	ö 00F6	ò 00F2	û 00FB	ù 00F9	İ 0130	Ö 00D6	Ü 00DC	ø 00F8	£ 00A3	Ø 00D8	Ş 015E	ş 015F
A-	á 00E1	í 00ED	ó 00F3	ú 00FA	ñ 00F1	Ñ 00D1	Ğ 011E	ğ 011F	¿ 00BF	® 00AE	¬ 00AC	½ 00BD	¼ 00BC	¡ 00A1	« 00AB	» 00BB
B-	░ 2591	▒ 2592	▓ 2593	│ 2502	┤ 2524	Á 00C1	Â 00C2	À 00C0	© 00A9	╣ 2563	║ 2551	╗ 2557	╝ 255D	¢ 00A2	¥ 00A5	┐ 2510
C-	└ 2514	┴ 2534	┬ 252C	├ 251C	─ 2500	┼ 253C	ã 00E3	Ã 00C3	╚ 255A	╔ 2554	╩ 2569	╦ 2566	╠ 2560	═ 2550	╬ 256C	¤ 00A4
D-	º 00BA	ª 00AA	Ê 00CA	Ë 00CB	È 00C8	€ 20AC	Í 00CD	Î 00CE	Ï 00CF	┘ 2518	┌ 250C	█ 2588	▄ 2584	¦ 00A6	Ì 00CC	▀ 2580
E-	Ó 00D3	ß 00DF	Ô 00D4	Ò 00D2	õ 00F5	Õ 00D5	µ 00B5		× 00D7	Ú 00DA	Û 00DB	Ù 00D9	ì 00EC	ÿ 00FF	¯ 00AF	´ 00B4
F-	- 00AD	± 00B1		¾ 00BE	¶ 00B6	§ 00A7	÷ 00F7	¸ 00B8	° 00B0	¨ 00A8	· 00B7	¹ 00B9	³ 00B3	² 00B2	■ 25A0	00A0