• 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))
```
• Write a Python script that computes the sum of squares from 1 to 150.

```python
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.

```plaintext
initialize sum=0
num=0

while num <= 150
    sum = sum + num**2
    num = num + 1

Display sum
```
While Loops

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

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

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

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.

Use colon after the logical expression.
While Loop

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

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

```python
sum = 0
um = 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

```python
sum = 0

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

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

```python
while (logical_expression):
    statement 1
    statement 2
    break
    statement 3
    statement 4
<statement outside the scope of while>
```
Difference of break and continue

- continue

```python
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
  – \( X = 3 \) \( \leftarrow \) scalar
  – \( X = [3, 5] \) \( \leftarrow \) 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
- $X=\text{‘TEST STRING’}$
- Elements of an array can be reached using indexing
  - $X[index]$ : Character at the location specified by index
  - index has to be integer!
    - $X[0]=\text{‘T’}$
    - $X[7]=? \quad \text{‘R’}$
Strings

- Indices start with 0
- Index 0 corresponds to the first letter of the string
- T=‘New String’
- T[0] = ? ‘N’
- Be careful 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 up to len(variable_name)-1
Strings

```python
>>> 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

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>T</td>
<td>E</td>
<td>S</td>
<td>T</td>
<td>S</td>
<td>T</td>
<td>R</td>
<td>I</td>
<td>N</td>
<td>G</td>
</tr>
</tbody>
</table>

• $X[-1] = \text{‘G’}$
• $X[-4] = \text{‘R’}$

• Negative indices start from -1 go down to -len(variable)
• Smaller indices than –len(variable) will be out of range and cause error
Strings

```python
>>> 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:
  
  \[ \text{index1} : \text{index2} \]

- 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

```python
>>> 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

```python
>>> 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

\[ \text{index1} : \text{index2} : \text{step\_size} \]

- 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

```python
>>> 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

```python
>>> T='This is a long test string'
>>> T[:5]
'This '
>>> 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

```python
>>> 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

```python
>>> str1="Yellow"
>>> str2='car'
>>> print str1+str2
Yellowcar
>>> str2='car'
>>> 3*str2
'carcarcar'
```
Examples

```python
>>> mystr='car'
>>> ('yellow'+mystr)*5
'yellowcaryellowcaryellowcaryellowcaryellowcar'

>>> 'yellow'+mystr*5
'yellowcarcarcarcarcar'

>>> 'yellow'*3+mystr*5
'yellowyellowyellowcaryellowcarcarcarcarcarcarcarcarcarcar'

>>> ('yellow'*3+mystr)*2
'yellowyellowyellowcaryellowyellowyellowyellowyellowyellowyellowcar'
```
‘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

```python
>>> 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>capitalize(s)</td>
<td>copy of s, with first character is capitalized</td>
</tr>
<tr>
<td>capwords(s)</td>
<td>copy of s, with first character of each word is capitalized</td>
</tr>
<tr>
<td>center(s,width)</td>
<td>center s in a field of given width</td>
</tr>
<tr>
<td>count(s,sub)</td>
<td>count the number of occurrences of substring sub in s</td>
</tr>
<tr>
<td>find(s,sub)</td>
<td>find the first position where sub occurs in s</td>
</tr>
<tr>
<td>join(list)</td>
<td>concatenate list of strings into one string</td>
</tr>
<tr>
<td>ljust(s,width)</td>
<td>left justify s in a field of given width</td>
</tr>
<tr>
<td>lower(s)</td>
<td>copy of s in all lowercase characters</td>
</tr>
<tr>
<td>lstrip(s)</td>
<td>copy of s with leading white space removed</td>
</tr>
</tbody>
</table>
# String Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>replace(s,oldsub,newsub)</code></td>
<td>replace occurrences of oldsub with newsub</td>
</tr>
<tr>
<td><code>rfind(s,sub)</code></td>
<td>like find but returns the right-most position</td>
</tr>
<tr>
<td><code>rjust(s,width)</code></td>
<td>right justify s in a field of given width</td>
</tr>
<tr>
<td><code>rstrip(s)</code></td>
<td>copy of s with trailing white space removed</td>
</tr>
<tr>
<td><code>split(s)</code></td>
<td>splits s into a list of substrings</td>
</tr>
<tr>
<td><code>upper(s)</code></td>
<td>copy of s with all characters converted to upper case letters</td>
</tr>
</tbody>
</table>
Examples

• You can use these functions in two different ways

```python
>>> 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

```python
>>> 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 functions can take arguments to refine their execution.
- 'find' method takes a second argument the index where it should start searching.
  ```python
  >>> str='Besiktas'
  >>> str.find('s',3)
  7
  ```
- Third argument is the index where find should stop searching.
  ```python
  >>> str.find('s',3,5)
  -1
  ```
Examples

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

```python
>>> 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

```python
>>> 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.

```python
>>> '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

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Htmlm</th>
<th>Char</th>
<th>Dec</th>
<th>Hx Oct</th>
<th>Htmlm</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 000</td>
<td>NUL</td>
<td>32</td>
<td>20 040</td>
<td>#32</td>
<td>Space</td>
<td>64</td>
<td>40 100</td>
<td>#64</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>33</td>
<td>21 041</td>
<td>#33</td>
<td>!</td>
<td>65</td>
<td>41 101</td>
<td>#65</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>34</td>
<td>22 042</td>
<td>#34</td>
<td>&quot;</td>
<td>66</td>
<td>42 102</td>
<td>#66</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>35</td>
<td>23 043</td>
<td>#35</td>
<td>#</td>
<td>67</td>
<td>43 103</td>
<td>#67</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>36</td>
<td>24 044</td>
<td>#36</td>
<td>$</td>
<td>68</td>
<td>44 104</td>
<td>#68</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>37</td>
<td>25 045</td>
<td>#37</td>
<td>%</td>
<td>69</td>
<td>45 105</td>
<td>#69</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>38</td>
<td>26 046</td>
<td>#38</td>
<td>&amp;</td>
<td>70</td>
<td>46 106</td>
<td>#70</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>39</td>
<td>27 047</td>
<td>#39</td>
<td>'</td>
<td>71</td>
<td>47 107</td>
<td>#71</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>40</td>
<td>28 050</td>
<td>#40</td>
<td>(</td>
<td>72</td>
<td>48 110</td>
<td>#72</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>TAB</td>
<td>41</td>
<td>29 051</td>
<td>#41</td>
<td>)</td>
<td>73</td>
<td>49 111</td>
<td>#73</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>012</td>
<td>LF</td>
<td>42</td>
<td>2A 052</td>
<td>#42</td>
<td>*</td>
<td>74</td>
<td>4A 112</td>
<td>#74</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>013</td>
<td>VT</td>
<td>43</td>
<td>2B 053</td>
<td>#43</td>
<td>+</td>
<td>75</td>
<td>4B 113</td>
<td>#75</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>014</td>
<td>FF</td>
<td>44</td>
<td>2C 054</td>
<td>#44</td>
<td>,</td>
<td>76</td>
<td>4C 114</td>
<td>#76</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>015</td>
<td>CR</td>
<td>45</td>
<td>2D 055</td>
<td>#45</td>
<td>-</td>
<td>77</td>
<td>4D 115</td>
<td>#77</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>016</td>
<td>SO</td>
<td>46</td>
<td>2E 056</td>
<td>#46</td>
<td>.</td>
<td>78</td>
<td>4E 116</td>
<td>#78</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>017</td>
<td>SI</td>
<td>47</td>
<td>2F 057</td>
<td>#47</td>
<td>/</td>
<td>79</td>
<td>4F 117</td>
<td>#79</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>020</td>
<td>DLE</td>
<td>48</td>
<td>30 060</td>
<td>#48</td>
<td>0</td>
<td>80</td>
<td>50 120</td>
<td>#80</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>021</td>
<td>DC1</td>
<td>49</td>
<td>31 061</td>
<td>#49</td>
<td>1</td>
<td>81</td>
<td>51 121</td>
<td>#81</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>022</td>
<td>DC2</td>
<td>50</td>
<td>32 062</td>
<td>#50</td>
<td>2</td>
<td>82</td>
<td>52 122</td>
<td>#82</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>023</td>
<td>DC3</td>
<td>51</td>
<td>33 063</td>
<td>#51</td>
<td>3</td>
<td>83</td>
<td>53 123</td>
<td>#83</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>024</td>
<td>DC4</td>
<td>52</td>
<td>34 064</td>
<td>#52</td>
<td>4</td>
<td>84</td>
<td>54 124</td>
<td>#84</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>025</td>
<td>NAK</td>
<td>53</td>
<td>35 065</td>
<td>#53</td>
<td>5</td>
<td>85</td>
<td>55 125</td>
<td>#85</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>026</td>
<td>SYN</td>
<td>54</td>
<td>36 066</td>
<td>#54</td>
<td>6</td>
<td>86</td>
<td>56 126</td>
<td>#86</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>027</td>
<td>ETB</td>
<td>55</td>
<td>37 067</td>
<td>#55</td>
<td>7</td>
<td>87</td>
<td>57 127</td>
<td>#87</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>030</td>
<td>CAN</td>
<td>56</td>
<td>38 070</td>
<td>#56</td>
<td>8</td>
<td>88</td>
<td>58 130</td>
<td>#88</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>031</td>
<td>EM</td>
<td>57</td>
<td>39 071</td>
<td>#57</td>
<td>9</td>
<td>89</td>
<td>59 131</td>
<td>#89</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>032</td>
<td>SUB</td>
<td>58</td>
<td>3A 072</td>
<td>#58</td>
<td>:</td>
<td>90</td>
<td>5A 132</td>
<td>#90</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>033</td>
<td>ESC</td>
<td>59</td>
<td>3B 073</td>
<td>#59</td>
<td>;</td>
<td>91</td>
<td>5B 133</td>
<td>#91</td>
<td>[</td>
</tr>
<tr>
<td>28</td>
<td>034</td>
<td>FS</td>
<td>60</td>
<td>3C 074</td>
<td>#60</td>
<td>&lt;</td>
<td>92</td>
<td>5C 134</td>
<td>#92</td>
<td>\</td>
</tr>
<tr>
<td>29</td>
<td>035</td>
<td>GS</td>
<td>61</td>
<td>3D 075</td>
<td>#61</td>
<td>=</td>
<td>93</td>
<td>5D 135</td>
<td>#93</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>036</td>
<td>RS</td>
<td>62</td>
<td>3E 076</td>
<td>#62</td>
<td>&gt;</td>
<td>94</td>
<td>5E 136</td>
<td>#94</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>037</td>
<td>US</td>
<td>63</td>
<td>3F 077</td>
<td>#63</td>
<td>?</td>
<td>95</td>
<td>5F 137</td>
<td>#95</td>
<td>_</td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com
Unicode

<table>
<thead>
<tr>
<th>Codepage 857 - Latin 5 (Turkey)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0-</td>
</tr>
<tr>
<td>1-</td>
</tr>
<tr>
<td>2-</td>
</tr>
<tr>
<td>3-</td>
</tr>
<tr>
<td>4-</td>
</tr>
<tr>
<td>5-</td>
</tr>
<tr>
<td>6-</td>
</tr>
<tr>
<td>7-</td>
</tr>
<tr>
<td>8-</td>
</tr>
<tr>
<td>9-</td>
</tr>
<tr>
<td>A-</td>
</tr>
<tr>
<td>B-</td>
</tr>
<tr>
<td>C-</td>
</tr>
<tr>
<td>D-</td>
</tr>
<tr>
<td>E-</td>
</tr>
<tr>
<td>F-</td>
</tr>
</tbody>
</table>