



Strings

Lecture 2

Assist. Prof. Dr. Abdul Hadi Mohammed

Strings

- **string**: A sequence of text characters in a program.
 - Strings start and end with quotation mark " or apostrophe ' characters.
 - Examples:

```
"hello"  
"This is a string"  
"This, too, is a string.    It can be very long!"
```
- A string may not span across multiple lines or contain a " character.

```
"This is not  
a legal String."  
"This is not a "legal" String either."
```
- A string can represent characters by preceding them with a backslash.
 - \t tab character
 - \n new line character
 - \" quotation mark character
 - \\ backslash character
 - Example: "Hello\tthere\nHow are you?"



Indexes

- Characters in a string are numbered with *indexes* starting at 0:

- Example:

```
name = "A. Alayen"
```

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----------|---|---|---|---|---|---|---|---|---|
| character | A | . | | A | l | a | y | e | n |

- Accessing an individual character of a string:

variableName [***index***]

- Example:

```
print (name, "starts with", name[0])
```

Output:

```
A. Alayen starts with A
```



String properties

- `len(string)`
 - number of characters in a string (including spaces)
- `str.lower(string)`
 - lowercase version of a string
- `str.upper(string)`
 - uppercase version of a string

- **Example:**

```
name = "Martin Douglas Stepp"
```

```
length = len(name)
```

```
big_name = upper()
```

```
print(big_name, "has", length, "characters")
```

Output:

```
MARTIN DOUGLAS STEPP has 20 characters
```



input

- `input` : Reads a string of text from user input.

- Example:

```
name = input("Howdy, pardner. What's yer name? ")  
print (name, "... what a silly name!")
```

Output:

```
Howdy, pardner. What's yer name? Paris Hilton  
Paris Hilton ... what a silly name!
```



Text processing

- **text processing**: Examining, editing, formatting text.
 - often uses loops that examine the characters of a string one by one
- A `for` loop can examine each character in a string in sequence.

- Example:

```
for c in "booyah":  
    print (c)
```

Output:

```
b  
o  
o  
y  
a  
h
```



Strings and numbers

- `ord(text)` - converts a string into a number.
 - Example: `ord("a")` is 97, `ord("b")` is 98, ...
 - Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.
- `chr(number)` - converts a number into a string.
 - Example: `chr(99)` is "c"
- **Exercise:** Write a program that performs a rotation cypher.
 - e.g. "Attack" when rotated by 1 becomes "buubdl"



ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---------|-----|------------------------|---------|-----|---------|---------|-----|------|---------|-----|-------|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | \$ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | (| 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 |) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [| 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D |] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

File processing

- Many programs handle data, which often comes from files.
- Reading the entire contents of a file:

```
        ("bankaccount.txt", "r", encoding="utf-8")      f:  
file_text = f.read()
```

: ensures the file is properly closed even if
there's an error.

: read mode (optional because it's the default).

: makes encoding explicit, good
practice for portability.



Line-by-line processing

- Reading a file line-by-line:

```
for line in open("filename") :  
    statements
```

Example:

```
count = 0  
with open("bankaccount.txt", "r", encoding="utf-8") as f:  
    for line in f:  
        count += 1  
print("The file contains", count, "lines.")
```

- **Exercise:** Write a program to process a file of DNA text, such as:
 ATGCAATTGCTCGATTAG
 - Count the percent of C+G present in the DNA.



Bank Account Management

Design a simple console-based banking application to manage customer accounts stored in a text file.

The application should include the following features:

1. Reading from and writing to files using Python.
2. Storing account data in dictionaries for efficient access and management.
3. Performing basic banking operations such as:
 - Depositing funds
 - Withdrawing funds
 - Checking account balances
4. Organizing code into functions for better readability and modularity.



File Format: bankaccount.txt

This file stores account data in CSV format:

```
AccountNumber,AccountHolder,AccountType,Balance  
1001,Ali Hassan,Savings,1500.0  
1002,Lina Ahmed,Checking,2300.5
```

Each row represents one account.



Load Accounts Function

```
def load_accounts(file_path):
    accounts = {}
    with open(file_path, 'r') as file:
        next(file)  # Skip the first line
        for line in file:
            parts = line.strip().split(',')
            if len(parts) != 4:
                print("Error in file format.")
                continue
            account_number, account_holder, account_type, balance = parts
            accounts[account_number] = {
                "AccountHolder": account_holder,
                "AccountType": account_type,
                "Balance": float(balance)
            }
    return accounts
```



Get, Deposit and Withdraw

```
def get_balance(accounts, account_number):  
    return accounts[account_number]["Balance"]  
  
def deposit(accounts, account_number, amount):  
    accounts[account_number]["Balance"] += amount  
  
def withdraw(accounts, account_number, amount):  
    if accounts[account_number]["Balance"] >= amount:  
        accounts[account_number]["Balance"] -= amount  
    else:  
        print("Insufficient funds.")
```



Save and Print Function

```
def save_accounts(file_path, accounts):
    with open(file_path, 'w') as file:
        file.write("AccountNumber,AccountHolder,AccountType,Balance\n")
        for account_number, details in accounts.items():
            file.write(f"{account_number},{details['AccountHolder']},
{details['AccountType']},{details['Balance']}\n")

def print_accouts(accounts):
    if accounts:
        print(f"{'AccountNumber':<15}{'AccountHolder':<20}
{'AccountType':<15}{'Balance':<10}")
        print("-" * 60)
        for account_number, details in accounts.items():
            print(f"{account_number:<15}{details['AccountHolder']:<20}
{details['AccountType']:<15}{details['Balance']:<10.2f}")
```



Example of usage App

```
file_path = "bankaccount.txt"
accounts = load_accounts(file_path)
print_accouts(accounts)
deposit(accounts, "1001", 100.0)
withdraw(accounts, "1001", 250.0)
save_accounts(file_path, accounts)
print_accouts(accounts)
```

