124- [PF] - 실습 - 함수를 사용하여 시저 암호 구현

함수를 사용하여 시저 암호 구현

실습 개요

프로그래밍에서 함수는 프로그램의 특정 태스크를 수행하는 이름이 지정된 섹션입니다. Python 에는 해당 언어에 의해 제공되는 `print()`와 같은 함수가 내장되어 있습니다. 또한 `import` 스테이트먼트를 통해 다른 개발자가 제공한 함수를 사용할 수도 있습니다. 예를 들어 `math.floor()` 함수를 사용하려면 `import math` 를 사용할 수 있습니다. Python 에서는 *사용자 정의 함수*라는 함수를 직접 만들 수 있습니다.

사용자 정의 함수와 관련된 논의를 촉진하려면 단순한 암호화 방식인 시저 암호를 구현하는 프로그램을 작성합니다. 시저 암호는 메시지의 문자를 취해 특정 수의 알파벳 위치에 따라 각 문자를 이동시킵니다.

본 실습에서는 다음을 수행합니다.

- 사용자 정의 함수 생성
- 여러 함수를 사용하여 시저 암호 암호화 프로그램 구현

예상 완료 시간

60 분

AWS Cloud9 IDE 액세스

1. 이 지침의 상단으로 이동한 다음 **Start Lab** 을 선택하여 실습 환경을 시작합니다.

   Start Lab 패널이 열리고 실습 상태가 표시됩니다.

2. *Lab status: ready* 라는 메시지가 표시되면 **X** 를 선택하여 **Start Lab** 패널을 닫습니다.
3. 지침의 맨 위에서 **AWS** 를 선택합니다.

새 브라우저 탭에서 AWS 관리 콘솔이 열립니다. 시스템에 자동으로 로그인됩니다.

> **참고:** 새 브라우저 탭이 열리지 않는 경우 일반적으로 브라우저에서 팝업 창을 열 수 없음을 나타내는 배너 또는 아이콘이 브라우저 상단에 표시됩니다. 배너 또는 아이콘을 선택하고 **Allow pop ups** 를 선택합니다.

4. AWS 관리 콘솔에서 **Services** > **Cloud9** 을 선택합니다. **Your environments** 패널에서 **reStart-python-cloud9** 카드를 찾아 **Open IDE** 를 선택합니다.

   AWS Cloud9 환경이 열립니다.

   > **참고:** *.c9/project.settings have been changed on disk* 라는 메시지가 담긴 팝업 창이 표시되면 **Discard** 를 선택하여 무시합니다. 마찬가지로, *Show third-party content* 라는 대화 창이 나타나면 **No** 를 선택하여 거절합니다.

Python 연습 파일 생성

5. 메뉴 모음에서 **File > New From Template > Python File** 을 선택합니다.

   이 작업은 제목이 없는 파일을 생성합니다.

6. 템플릿 파일에서 샘플 코드를 삭제합니다.
7. **File > Save As...**를 선택하고, 연습 파일에 적절한 이름(예: *caesar-cipher.py*)을 입력한 다음 **/home/ec2-user/environment** 디렉터리에 저장합니다.

터미널 세션에 액세스

8. AWS Cloud9 IDE 에서 **+** 아이콘을 선택하고 **New Terminal** 을 선택합니다.

   터미널 세션이 열립니다.

9. 현재 작동 중인 디렉터리를 표시하려면 `pwd` 를 입력합니다. 이 명령은 **/home/ec2-user/environment** 를 가리킵니다.
10. 이 디렉터리에서 이전 섹션에서 생성한 파일을 찾습니다.

연습 1: 사용자 정의 함수 생성

Python 에서 시저 함수를 구현하는 프로세스를 시작하려면 단순한 사용자 정의 함수를 생성해야 합니다.

11. IDE 의 탐색 창에서 이전 *Python 연습 파일 생성* 섹션에서 생성한 파일을 선택합니다.
12. 다음과 같이 문자열 인수를 취하고 주어진 문자열을 자신과 연결 또는 결합하는 `getDoubleAlphabet` 이라는 함수를 정의합니다.

```
def getDoubleAlphabet(alphabet):
    doubleAlphabet = alphabet + alphabet
    return doubleAlphabet
```

> **참고:** 함수 스테이트먼트에서 필수인 부분은 `def` 키워드와 이름, 그리고 콜론(:)입니다. 또한 Python 에서 변수를 선언할 필요가 없으며 해당 데이터 유형은 할당 스테이트먼트에서 추론됩니다.

13. 파일을 저장합니다.
14. 함수의 기능을 이해하기 위해 샘플 입력 `alphabet="ABC"`를 예로 들어 보겠습니다. 이 입력의 반환 문자열은 `"ABC" + "ABC" = "ABCABC"`입니다. 더하기 기호(+)는 여러 문자열을 하나의 문자열로 연결합니다.

다음 연습을 통해 단순한 태스크를 수행하는 더 많은 함수를 정의합니다. 그런 다음 이러한 함수를 결합하여 시저 암호 프로그램을 만듭니다.

연습 2: 메시지 암호화

정의할 다음 함수는 사용자에게서 암호화할 메시지를 요청합니다. 내장된 `input()`이라는 함수를 사용합니다.

15. 텍스트 편집기에 다음 코드를 입력하고 파일을 저장합니다.

```
def getMessage():
    stringToEncrypt = input("Please enter a message to encrypt: ")
    return stringToEncrypt
```

참고: 함수는 특정 태스크를 수행해야 합니다. 일반적으로 함수는 특정 태스크를 수행하기 때문에 이 연습의 함수도 길이가 짧을 것입니다. 이 함수는 문자열을 반환하지만 `getDoubleAlphabet()` 함수와 같이 인수를 취하지 않습니다.

연습 3: 암호 키 가져오기

*암호 키*는 문자를 얼마나 멀리 이동할지 결정합니다. 2 개의 알파벳을 사용하여 1~25 사이의 모든 정수인 암호 키를 가질 수 있습니다. 인덱스 26 에서 키를 세면 원본 메시지로 돌아갔을 때 키가 다시 돌아가기 때문에 그렇게 해서는 안 됩니다.

16. 다음 코드를 입력하여 함수를 정의하여 사용자에게 암호 키를 요청합니다.

```
def getCipherKey():
    shiftAmount = input( "Please enter a key (whole number from 1-25): ")
    return shiftAmount
```

17. 파일을 저장합니다.

연습 4: 메시지 암호화

지금까지 살펴본 함수는 짧고 단순했습니다. 특정 태스크 내에서만 함수를 유지하는 경우에는 이것이 일반적입니다. `encryptMessage` 함수는 비교적 깁니다.

18. 코드를 작성하기 전에 다음과 같이 암호화에 대한 알고리즘을 계획해야 합니다.
    1. 메시지, 암호 키, 알파벳이라는 인수 3 개를 취합니다.
    2. 변수를 초기화합니다.
    3. `for` 루프를 사용하여 메시지의 각 문자를 트래버스합니다.
    4. 특정 문자의 위치를 찾습니다.
    5. 암호 키를 고려하여 특정 문자의 새 위치를 결정합니다.
    6. 현재 문자가 알파벳으로 되어 있으면 새 문자를 암호화된 메시지에 추가합니다.
    7. 현재 문자가 알파벳으로 되어 있지 않으면 현재 문자를 추가합니다.
    8. 메시지의 모든 문자를 소진한 후에 암호화된 메시지로 돌아갑니다.
19. 연습 파일에 다음 코드를 입력하고 이전 알고리즘의 단계를 검토하여 논리를 따릅니다.

```
def encryptMessage(message, cipherKey, alphabet):
```

```
    encryptedMessage = ""
    uppercaseMessage = ""
    uppercaseMessage = message.upper()
    for currentCharacter in uppercaseMessage:
        position = alphabet.find(currentCharacter)
        newPosition = position + int(cipherKey)
        if currentCharacter in alphabet:
            encryptedMessage = encryptedMessage + alphabet[newPosition]
        else:
            encryptedMessage = encryptedMessage + currentCharacter
    return encryptedMessage
```

20. 파일을 저장합니다.

연습 5: 메시지 암호화 해제

함수는 다시 사용할 수 있기 때문에 유용합니다. `encryptMessage()` 함수를 다시 사용하여 `decryptMessage()` 함수를 작성합니다. 이 단순한 암호화에서는 각 문자를 다시 이동하는 대신 암호화를 실행 취소할 수 있습니다. 즉, 암호 키를 추가하는 대신 암호 키를 뺍니다. 대부분의 논리를 다시 작성하는 것을 피하기 위해 부정(negative) 암호 키를 전달합니다.

21. 다음으로, 다음 코드를 입력하고 파일을 저장합니다.

```
def decryptMessage(message, cipherKey, alphabet):
    decryptKey = -1 * int(cipherKey)
    return encryptMessage(message, decryptKey, alphabet)
```

연습 6: 기본 함수 생성

시저 암호 프로그램을 작성하는 데 도움이 되는 사용자 정의 함수 모음을 작성했습니다. 프로그램의 기본 논리는 당연히 함수에도 포함되어 있습니다.

22. 코드를 살펴보기 전에 논리를 계획합니다.

1. 영문 알파벳을 포함하는 문자열 변수를 정의합니다.
2. 문자를 이동하려면 알파벳 문자열을 두 배로 늘립니다.
3. 사용자에게서 암호화할 메시지를 가져옵니다.
4. 사용자에게서 암호 키를 가져옵니다.
5. 메시지를 암호화합니다.
6. 메시지의 암호화를 해제합니다.

23. 연습 파일에 다음 코드를 입력하고 이전 알고리즘의 단계를 검토하여 논리를 따릅니다.

```
def runCaesarCipherProgram():
    myAlphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    print(f'Alphabet: {myAlphabet}')
    myAlphabet2 = getDoubleAlphabet(myAlphabet)
    print(f'Alphabet2: {myAlphabet2}')
    myMessage = getMessage()
    print(myMessage)
    myCipherKey = getCipherKey()
    print(myCipherKey)
    myEncryptedMessage = encryptMessage(myMessage, myCipherKey, myAlphabet2)
    print(f'Encrypted Message: {myEncryptedMessage}')
    myDecryptedMessage = decryptMessage(myEncryptedMessage, myCipherKey,
myAlphabet2)
    print(f'Decypted Message: {myDecryptedMessage}')
```

디버깅 및 프로그램의 이해에 도움을 주기 위해 `print()` 스테이트먼트가 추가되었지만, 프로그램이 제대로 작동하기 위해 반드시 필요한 것은 아닙니다.

24. 파일을 저장하고 실행한 다음 결과를 확인합니다.

아무 일도 일어나지 않습니다. 왜일까요? 이번 주 Y 씨의 과제는 다소 길고, 일부 파일 관리와 네트워크 도구 세트가 필요합니다. 함수를 호출하지 않았습니다.

25. 함수를 호출하려면 **.py** 파일에 다음 줄을 추가하고 파일을 저장합니다.

```
runCaesarCipherProgram()
```

26. 프로그램을 다시 실행합니다. 출력은 다음과 비슷해야 합니다.

```
Alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
Alphabet2: ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
Please enter a message to encrypt: new message
new message
Please enter a key (whole number from 1-25): 4
4
Encrypted Message: RIA QIWWEKI
Decypted Message: NEW MESSAGE
```

27. 다른 입력으로 프로그램을 다시 실행합니다.

축하합니다! 사용자 정의 함수를 사용하여 작업하고 암호화 프로그램을 구현했습니다!

실습 종료

축하합니다! 실습을 마치셨습니다.

28. 이 페이지의 상단에서 **End Lab** 을 선택한 다음 Yes 를 선택하여 실습 종료를 확인합니다.

    *DELETE has been initiated... You may close this message box now.*라는 내용의 패널이 표시됩니다.

29. *Ended AWS Lab Successfully* 라는 메시지가 잠시 표시되어 실습이 종료되었음을 나타냅니다.

추가 리소스

AWS Training and Certification 에 대한 자세한 내용은 https://aws.amazon.com/training/을 참조하십시오.

여러분의 피드백을 환영합니다. 제안이나 수정 사항을 공유하려면 AWS Training and Certification Contact Form 에서 세부 정보를 제공해 주십시오.

© 2022 Amazon Web Services, Inc. 및 계열사. All rights reserved. 본 내용은 Amazon Web Services, Inc.의 사전 서면 허가 없이 전체 또는 일부를 복제하거나 재배포할 수 없습니다. 상업적인 복제, 대여 또는 판매는 금지됩니다.