Command Line Arguments / Slicing

1 Argparse

A convenience package for handling command line parameters

In [ ]:
```
# Simulate argv because notebooks have their very own stuff there

import sys
sys.argv = ["program.py", "-x1"]
sys.argv = ["program.py", "-x", "1"]  # argv are always strings
```

1.1 Basic workflow

In [ ]:
```
# step 0: import the package
import argparse

# Step 1: instantiate a parser
arg_parser = argparse.ArgumentParser()

# step 2+: Add our parameters
arg_parser.add_argument("-x", type=int)

#step 3: Parse the sys.argv (does use sys.argv as default)
parsed_params = arg_parser.parse_args()

#step 4: extract the variables
cmd_params = vars(parsed_params)

print(cmd_params)
```

1.2 Further important tricks

1.2.1 Long and short forms

In [ ]:
```
arg_parser = argparse.ArgumentParser()
arg_parser.add_argument("-f", "--first", type=int)
arg_parser.add_argument("-s", "--second", type=str) # note the second will be a string, even if we provide a number


sys.argv = ["program.py", "-f", "1", "-s", "2"]
cmd_params = vars(arg_parser.parse_args())
print(cmd_params)

# the name is always defined by -- name
```

1.2.2 default values

In [ ]:
```
arg_parser = argparse.ArgumentParser()
arg_parser.add_argument("-x", type=int, default=1337)

sys.argv = ["program.py", "-x", "1"]
# sys.argv = ["program.py"] # ccomment in to test defaults
cmd_params = vars(arg_parser.parse_args())
print(cmd_params)
```

1.2.3 existence checks

In [ ]:
```
arg_parser = argparse.ArgumentParser()
arg_parser.add_argument("--run", action="store_true")

sys.argv = ["program.py", "--run"]
# sys.argv = ["program.py"]
cmd_params = vars(arg_parser.parse_args())
print(cmd_params)

# The param will always be there and you get True of False to see if it was used in the command line
```

In [ ]:
```
sys.argv = ["program.py", "-h"]
# sys.argv = ["program.py"]
cmd_params = vars(arg_parser.parse_args())
print(cmd_params)
```

2 Slicing

Access specific parts of an indexed object

- Lists
- Tuples
- Strings (they are nothing but a list of characters)

Access with []
Define ranges with [x:y]
Define step sizes with [x:y:stepsize]
Always makes a copy (not a reference)

In [ ]:
```
#create list to work with
tmp_list = [0,1,2,3,4,5,6,7,8,9] # for lists
tmp_list = "0123456789"         # for strings
```

In [ ]:
```
# Access the second element
print(tmp_list[1])

# Remember lists start with element 0
```

In [ ]:
```
# Access the second to last element
print(tmp_list[-2])

# Here -1 is actually the very last because -0 does not exist
```

In [ ]:
```
# Sublist from second to second to last
print(tmp_list[2:-2])

# if the first index is larger than the second (eg: tmp_list[3:2]) you get an empty list
```

In [ ]:
```
# Sublist from second to second to last but only taking every second element
print(tmp_list[2:-2:2])
```

In [ ]:
```
# Get the list from the second one onwards
print(tmp_list[2:])
```

In [ ]:
```
# Get the list until the second to last
print(tmp_list[:-2])
```

2.2 Complex Example: Hangman

In [ ]:
```
min_word_len = 6 # minimum length of the target word

# Load words into a set
with open('words.txt', 'r') as words_file:
    words = [l.strip().lower() for l in words_file if len(l) >= min_word_len] #do many things in one pretty line

print(len(words)) # Print the number of words with that critera
```

In [ ]:
```
import random # package for random numbers
selected_word = random.choice(words) # pick one random word
```

In [ ]:
```
print(selected_word) # cheater
```

In [ ]:
```
correct_letters = [False] * len(selected_word)
guessed_letters = []
current_word = ['_'] * len(selected_word)
wrong_letters_count = 0

while wrong_letters_count < 7 and False in correct_letters:
    # ask the user to enter a letter
    letter = input('Guess a letter: ')
    letter = letter.lower()
    # only one letter at a time!
    while (len(letter) is not 1):
        letter = input('Guess one letter only: ')
        letter = letter.lower()


    if letter not in guessed_letters:
        # the word contain the player's letter!
        if letter in selected_word:

            # now where do we find the letter?
            # We're not using find() because it only finds the first occurrence
            for i in range(len(selected_word)):
                if selected_word[i] == letter:
                    # this letter was guessed correctly,
                    # so let's set that position to True
                    correct_letters[i] = True

                    # show the player the current status by showing all
                    # correct letters in the word
                    current_word[i] = selected_word[i]
                    # the known letters are in a list, so to print them correctly,
                    # they need to be joined to a string
                    print('This is what you know so far: ' + ' '.join(current_word))

        else:
            # that means one attempt less available...
            wrong_letters_count += 1
            print('Sorry, my word does not contain ' + letter + '. You have ' + str(7 - wrong_letters_count + 1) + ' guesses remaining.')

            # this letter cannot be used anymore
            guessed_letters.append(letter)


if False in correct_letters:
    print('Sorry, you lost.')
    # everything was guessed correctly
else:
    print('Yay, you won!')
```