

# Reinforcement

Andey Robins & Dr Borowczak

October 11, 2022

## Reinforcement

# Outline

- ▶ for loop setups
- ▶ Example List Problem
- ▶ Text Adventure Updates

for Loops

# Building Structures

```
sentence = "someone is good"
translation = ""
for word in sentence.split():
    translation += en2tp[word]
    translation += " "
print(translation)
```

*# first loop*

translation == "jan "

*# second loop*

translation == "jan li "

*# third loop*

translation == "jan li pona "

## Takeaway

When we move through one collection, we can build another collection simultaneously.

## Another Example

Given a piece of text, count the frequency of each letter.



# Setup

We will use an array with 26 spots to count the Latin letter frequency.

`frequency[0]` *# count of 'a'*

`frequency[3]` *# count of 'd'*

`frequency[25]` *# count of 'z'*

```
letter_freq = []  
idx = 0  
while idx < 26:  
    letter_freq.append(0)  
    idx += 1
```

```
letter_freq == [0, 0, 0, 0, ..., 0]
```

# Iteration

This prints out each individual word.

```
for word in text.split():  
    print(word)
```

This prints out each individual character.

```
for word in text.split():  
    for char in word:  
        print(char)
```

# Nested for Loops

Nested for loops allow us to look through a collection that is inside a collection.

- ▶ We can look at characters in a string, which are in a list.
- ▶ We can look at numbers in a list, which are in a list.
- ▶ We can look at strings in a list, which is in a list, which is in a list, ....

```
for word in text.split():  
    for char in word:  
        print(ord(char)-97)
```

```
>>> ord('a')
```

```
97
```

```
>>> ord('a')-97
```

```
0
```

```
>>> ord('b')
```

```
98
```

```
>>> ord('b')-97
```

```
1
```



```
for word in text.split():  
    for char in word:  
        letter_freq[ord(char)-97] += 1
```

## Iterative Output

```
idx = 0
while idx < 26:
    print(chr(idx+97), "-", letter_freq[idx])
    idx += 1
```

$$a = 12$$

$$b = 6$$

...

$$y = 2$$

$$z = 0$$

## Example Problem

## Example Problem

Given a list, write a function which returns a list without duplicates.

Possible Approaches?

## Hint

Think in terms of building a collection step by step.

If I only had two items, how could I remove duplicates?



Save the first item, then if the second item is different, add it too.

Now, extend this to three items. Assuming I process the first two properly, how do we know if a list of three has duplicates?

If there's only one item, compare the third item to it. If there are two, compare the third item to the first item and then the second.

Therefore, if we have a list of all the unique elements, and we have a new element, we can compare the new element to each other element. If we don't see a duplicate, we can add the new element to the list. Repeat this process until we have no more new elements.

Outline our function

```
def remove_dupes(input_list):  
    output_list = []  
  
    return output_list
```

Create the nested loops

```
def remove_dupes(input_list):  
    output_list = []  
    for potential_item in input_list:  
        for item in output_list:  
            ...  
    return output_list
```

Track if we found the item

```
def remove_dupes(input_list):  
    output_list = []  
    for potential_item in input_list:  
        item_found = False  
        for item in output_list:  
            ...  
    return output_list
```

Check if the item is unique

```
def remove_dupes(input_list):  
    output_list = []  
    for potential_item in input_list:  
        item_found = False  
        for item in output_list:  
            if item == potential_item:  
                item_found = True  
  
    return output_list
```



After checking all the saved items, if it hasn't been found, it's unique! So save it!

```
def remove_dupes(input_list):  
    output_list = []  
    for potential_item in input_list:  
        item_found = False  
        for item in output_list:  
            if item == potential_item:  
                item_found = True  
  
        if not item_found:  
            output_list.append(potential_item)  
  
    return output_list
```

## Game Modifications

# What to do?

We want to add an increase in difficulty to the game as you progress.  
The simplest way I see to do this, we have harder monsters!

```
MONSTER_HEALTH = [10, 10, 10, 15, 15, 20, 20, 20, 25, 40]
```

Then, we can prevent the player from moving or resting if there are monsters nearby!

```
if MONSTER_HEALTH[ROOM] > 0:  
    print("There's a monster to kill first!")  
    return
```

# The End

How can we create an end condition with our game now that we're storing monster health in a list?

```
if ROOM > len(MONSTER_HEALTH):  
    print("You finally escape into the light. Free at last!")  
    CONTINUE = False  
    return
```

# Interesting Loot

Create some item descriptions.

```
INVENTORY = []
```

```
LOOT = ["A Magic Sword", "A cryptic scroll",  
        "A leather chord", "A Potion of Healing",  
        "A Bottle of Water"]
```



Loot drop

```
global PLAYER_GOLD, INVENTORY, LOOT
rand = random.randrange(5)
if rand % 2 == 0:
    print("Found some gold!")
    PLAYER_GOLD += 2
elif rand == 1:
    print("What's this?")
    # cool loot
else:
    print("Found nothing.")
```

If we find cool loot, pop an item from the Loot list and add it to the player's inventory.

```
looted_item = LOOT.pop()  
print("Found", looted_item)  
INVENTORY.append(looted_item)
```

Add an inventory command

```
def act(action):  
    ...  
    elif action == "I" or action == "i":  
        inventory()  
    ...
```

## Requirements for `inventory()`

- ▶ Tell the player which items they have
- ▶ Behave nicely if there are no items
- ▶ Behave nicely when there is an arbitrary number of items

```
def inventory():  
    if "Magical Sword" in INVENTORY:  
        print("You have a magical sword")  
    ...
```

```
def inventory():  
    for item in INVENTORY:  
        print("You have", item)
```

## Bounds Checking

```
def inventory():  
    if len(INVENTORY) == 0:  
        print("You haven't found any items.")  
        return  
  
    for item in INVENTORY:  
        print("You have", item)
```

Now we can also make our loot function check if there are more items to find.

```
if len(LOOT) != 0:
    looted_item = LOOT.pop()
    print("Found", looted_item)
    INVENTORY.append(looted_item)
else:
    print("Found nothing.")
```



# Extensions

- ▶ Provide a way of playing an endless mode.
- ▶ Provide a way for the player to increase their damage.
- ▶ Provide a way for the player to backtrack.
- ▶ Add special loot that is in certain rooms.

## Questions