Topic 6: Loops

David H Smith IV

University of Illinois Urbana-Champaign

Wed, Sep 22 2021

Enumerate

Poll Question: Unpacking

What are the values of foo?

```
1 foo, bar = (1, 2)
```

A. 1
B. 2
C. (1, 2)
D. Error

Poll Question: Unpacking

What is the value of z?

```
1 x, y, z = [22, [33, 44], [66]]
```

A. 22
B. 33
C. `[33, 44]`
D. `[66]`

Poll Question: Enumerate

What is the value of `x` after this code runs?

```
1 orig_list = ["I", "am", "Groot"]
2 x = enumerate(orig_list)
```

A. Error
B. `[(1, "I"), (2, "am"), (3, "Groot")]`
C. `[(0, "I"), (1, "am"), (2, "Groot")]`
D. `[[0, "I"], [1, "am"], [2, "Groot"]]`
E. Something else

Poll Question: Enumerate

What is the value of `x` after this code runs?

```
1 orig_list = ["I", "am", "Groot"]
2 x = enumerate(orig_list)
```

A. Error
B. `[(1, "I"), (2, "am"), (3, "Groot")]`
C. `[(0, "I"), (1, "am"), (2, "Groot")]`
D. `[[0, "I"], [1, "am"], [2, "Groot"]]`
E. Something else

Enumerate is like `range()`. On it's own it's just an object that we can iterate over.

Poll Question: Enumerate

What is the value of `y` after this code runs?

```
1 orig_list = ["I", "am", "Groot"]
2 x = enumerate(orig_list)
3 y = list(x)
```

A. Error

B. `[(1, "I"), (2, "am"), (3, "Groot")]`

C. `[(0, "I"), (1, "am"), (2, "Groot")]`

D. `[[0, "I"], [1, "am"], [2, "Groot"]]`

Poll Question: Enumerate

For this code, what is the varible `type` of item at each iteration?

```
1 orig_list = [3, 7, 22, 90]
2 for item in enumerate(orig_list):
3   print(item)
```

A. `tuple`
B. `list`
C. `int`
D. This code has an error

Poll Question: Enumerate

```
for item in enumerate(x):
  print(item)
```

+

```
i, val = (0, 2)
```

=

```
for i, val in enumerate(x):
  print(i, val)
```

Poll Question: Enumerate

What is the contents of new list?

```
1 orig_list = [3, 7, 22, 90]
2 new_list = []
3 for index, value in enumerate(orig_list):
4   if (index % 2) == 0:
5     new_list.append(value)
```

A. [3, 7]

B. [3, 22]

C. [3, 7, 22, 90]

D. [7, 90]

Why?!?

Why would we ever want to use this enumerate?

Consider the Following

I want to replace all even numbers in a list with the word "Even" and odd numbers with the word "Odd". Will this code do it?

```
1 x = [1, 2, 3, 4]
2 for item in x:
3   item = "Even" if item % 2 == 0 else "Odd"
```

- A Yes :D
- B No D:

Consider the Following

I want to replace all even numbers in a list with the word "Even" and odd numbers with the word "Odd". Will this code do it?

```
1 x = [1, 2, 3, 4]
2 for item in x:
3   item = "Even" if item % 2 == 0 else "Odd"
```

- A Yes :D
- B No D:

How could we accomplish this task?

Consider the Following

Will this work?

```
x = [1, 2, 3, 4]
for i, item in enumerate(x):
  x[i] = "Even" if item % 2 == 0 else "Odd"
```

Now Side by Side

1) Doesn't update the list because `item` is a different variable that simply references a value in the list. Setting it equal to a new value only updates the value it is references and doesn't change the original list.

```
x = [1, 2, 3, 4]
for item in x:
  item = "Even" if item % 2 == 0 else "Odd"
```

2) This directly references the list via the index `i` and updates the value in the list.

```
x = [1, 2, 3, 4]
for i, item in enumerate(x):
  x[i] = "Even" if item % 2 == 0 else "Odd"
```

Break vs Continue

Poll Question: Break

How many chars are printed?

```
for c in "sleepy":
  if c == "e":
    break
  print(c)
```

A. 4
B. 1
C. 2
D. 6

Poll Question: Break vs Return

On which inputs do these functions behave differently?

```
1 def func(a_list):
2   for item in a_list:
3     if item == "":
4       break
5     print(item)
6   print("done")
```

```
1 def func(a_list):
2   for item in a_list:
3     if item == "":
4       return
5     print(item)
6   print("done")
```

- (A) `func(["a", "b", "", "d"])`
- (B) `func(["a", "b", "c", ""])`
- (C) both
- (D) neither

Poll Question: Continue

How many items are printed?

```
1 mixed_list = ['hi', '3', math.pi, 'there', ['CS', 437]]
2 for item in mixed_list:
3   if type(item) != str:
4     continue
5   print(item)
```

A. 0
B. 3
C. 2
D. 6

Break vs Return vs Continue

- **continue** →Skips everything below it and goes back to the beginning of the loop.
- **return** →Leaves function with return value.
- **break** →Exits loop it is apart of.

When would we want to use these?

1. When would we want to use continue?

When would we want to use these?

1. When would we want to use continue?
    1. Concatenating all strings in a list of things.
    2. Summing all integers/floats in a list of things.
    3. Validating user input.

When would we want to use these?

1. When would we want to use continue?
   1. Concatenating all strings in a list of things.
   2. Summing all integers/floats in a list of things.
   3. Validating user input.
2. When would we want to use break?

When would we want to use these?

1. When would we want to use continue?
   1. Concatenating all strings in a list of things.
   2. Summing all integers/floats in a list of things.
   3. Validating user input.
2. When would we want to use break?
   1. Searching for the occurance of an item that meats a condition in the list.
   2. Exiting an otherwise infinite loop when the user wants to exit.

Poll Question: Summing Nums in List

What should we replace the question marks with?

```
1 def sum_nums(x):
2   s = 0
3   for item in x:
4     if type(item) != int or float:
5       ???
6     s += item
```

A. break
B. continue
C. There's another issue with this code

Poll Question: Summing Nums in List

What should we replace the question marks with?

```
1 def sum_nums(x):
2   s = 0
3   for item in x:
4     if type(item) != int or float:
5       ???
6     s += item
```

A. break
B. continue
C. There's another issue with this code

What should we replace this line of code with?

Poll Question: Summing Nums in List

What should we replace the question marks with?

```
1 def sum_nums(x):
2   s = 0
3   for item in x:
4     if type(item) != int and type(item) != float:
5       ???
6     s += item
```

A. break

B. continue

C. There's another issue with this code

Poll Question: Summing Nums in List

What should we replace the question marks with?

```
def sum_nums(x):
  s = 0
  for item in x:
    if type(item) != int and type(item) != float:
      ???
    s += item
```

A. break
B. continue
C. There's another issue with this code

Key Takeaway: Break leaves the loop. Continue skips to the next iteration.

General Loop Practice

Task: Validate User Input

Problem Statement: Create a function that gets 10 words that contain the letter "e", stores them in a list, then returns them. Note that this problems uses nested loops but not break or enumerate.

Task: Validate User Input

Problem Statement: Create a function that gets 10 words that contain the letter "e", stores them in a list, then returns them. Note that this problems uses nested loops but not break or enumerate.

```
1 def no_e():
2   l = []
3   for i in range(0, 10):
4     word = input("Enter a word with the letter e: ")
5     while "e" not in word:
6       word = input("Enter a word with the letter e: ")
7     l.append(word)
8   return l
```

Task: Validate User Input

Problem Statement: Create a function that keeps asking the user for strings of an even length and adding them to a list until the user enters a string of an odd length. Then return the final list. You'll want to use a "while True:" loop here.

Task: Validate User Input

Problem Statement: Create a function that keeps asking the user for strings of an even length and adding them to a list until the user enters a string of an odd length. Then return the final list. You'll want to use a "while True:" loop here.

```
1 def get_even_words():
2   l = []
3   while True:
4     user_in = input("Enter a word with an even number of vowels: ")
5     if len(user_in) % 2 != 0:
6       print("That word has an odd number of letters. Terminating!!")
7       break
8     l.append(user_in)
```