

Loops

Tuples

Named Tuples

Loops:

Repetition without being repetitive

Syntax

```
for <item> in <collection>:  
    # Code to loop over  
    print("Repeat this")
```

Example: Iterating through collections

- Print each character of a string
- Print each element of a list
- Print each element of a tuple

Concept Test

- What is the output of this code?

```
for x in [1, 2, 3]:  
    print('Hello'*x) # using x inside the loop
```

- A. 1 2 3
- B. 'Hello' is printed 3 times
- C. Hello
HelloHello
HelloHelloHello
- D. None of the above

Range() function

- Used in a loop when we know the number of times we want to repeat executing some code

```
range(5)      # think of it as producing a list [0, 1, 2, 3, 4]
range(1, 5) # The first parameter is a starting value
            # The second parameter is the stopping value
            # Output [1, 2, 3, 4]
range(0, 10, 2) # The third parameter is the step count
                 #[0, 2, 4, 6, 8]

for x in range(5):
    print('Hello')
```

Concept Test

What is the output of this code?

```
for x in range(1,4,2):  
    print(2**x, end = " ")
```

- A. 1 4 2
- B. 2 16 4
- C. 2 8 16
- D. 2 8
- E. None of the above

The accumulator pattern

Useful for "accumulating" something while going through a collection.

Example: sum the elements of a list of integers

```
def sumList(lst):  
    ''' return the sum of elements in lst  
    ...
```

The accumulator pattern

Useful for "accumulating" something while going through a collection.

Example: find the number of a times a vowel occurs in a word

```
def numVowels(word):
    ''' returns the number of vowels
        in a word '''
```

The accumulator pattern

Useful for "accumulating" something while going through a collection.

We can accumulate into a list

```
def powerOfTwo(lst):
    ''' returns a new list: 2**lst
    '''
```

The accumulator pattern

Useful for calculating something from repeated smaller computations

Example: find the sum of a series

```
def sumGeometric series(n):  
    '''returns the sum of the series  
    1 + 2**1 + 2**2 + 2**3 + ...+ 2**n'''  
    Assume n>=0 '''
```

Conditionals: if, if-else

```
# Assume x exists and represents an exam score  
if x > 70:  
    print ("Grade = Passed")
```

```
#####
```

```
if x > 70:  
    print ("Grade = Passed")  
else:  
    print ("Grade = Failed")
```

Conditionals: if-elif-else

```
if x>90:  
    print ("Grade = A")  
elif x >80:  
    print ("Grade = B")  
elif x >70:  
    print ("Grade = C")  
elif x >60:  
    print ("Grade = D")  
else:  
    print("Grade = F")
```

Loops (accumulator pattern and conditionals)

```
def countOddNumbers(lst):  
    ''' returns the number of odd numbers in lst  
    ...
```

Loops and conditionals

```
def containsOddNumber(lst):  
    "return True if any element in lst is odd,  
    otherwise return False"
```

```
def containsAllOdd(lst):  
    "return True if all the elements in lst are odd,  
    otherwise return False"
```

Concept Test

```
def containsOddNumber(lst):
    '''return True if any element in lst is
    odd, otherwise return False'''
    for x in lst:
        if (x % 2 == 1):
            return True
        else
            return False
```

Is the above implementation correct? (Why or Why not)

- A. Yes
- B. No

More on the accumulator pattern

```
def countWords(sentence):  
    "returns the number of words in the sentence"
```

```
def countWords(sentence, len):  
    "returns the number of words in the  
    sentence with length greater than len"
```

Tuples

- Similar to lists: store a sequence of elements

```
lst = [ 10, 20] //ex of a list
```

```
tup = (10, 20) //ex of a tuple
```

- Elements are ordered and can be accessed using the appropriate index

```
tup[0]
```

```
tup[1]
```

- Different from lists in the following ways
 - Can't change an element in the tuple
 - Can't sort the elements in a tuple

Named Tuples

- Used to package data with multiple attributes: e.g. representing a student in your program
- A student's attributes may be: name, perm number, major etc.
- Named tuples make it easier to access each attribute

```
from collections import namedtuple
```

```
#Design your named tuple object
```

```
Student = namedtuple('Student', 'name perm major gpa')
```

```
# Create objects of type Student
```

```
s1 = Student("Jack", 123443, CS, 3.8)
```

```
s2 = Student("Mary", 8932737, CE, 3.9)
```

```
# Access the elements of the objects
```

```
print(s1.name, s1.perm)
```