Tuples, for-loops, range, indexes, slices

CS195 - Lectures 6 and 7
Instructor: Dr. V

- tuples
    - assignment syntax
    - immutable
- packing, unpacking in assignments
- len
- in
- +, +=, *, *=
- max, min, sum
- for loop

Tuples

- a tuple is a finite ordered sequence of items
- a tuple is used to store multiple items into a single variable
- python tuples are immutable

```
# tuple syntax: comma-separated items, in parentheses
mytuple1 = ("apple", "banana", "cherry")
# in most cases, you can skip the parentheses
mytuple2 = "apple", "banana", "cherry"
```

Automatic unpacking

- assigning multiple variables to a tuple automatically unpacks the tuple into multiple elements

```
mytuple1 = "apple", "banana", "cherry"
a, b, c = mytuple1

x, y = 7, 8
```

Length of a sequence

- use len(*t*) to find out length of sequence *t*
    - works with both tuples and strings

```
>>> mytuple1 = "apple", "banana", "cherry"

>>> len( mytuple1 )
??

>>> len( "apple" )
??
```

Checking if something is in a sequence

- use the **in** operator to check if item is in sequence
  - works with both tuples and strings
    - with strings (not tuples) you can even check whether a sequence of characters is inside your string

```
>>> mytuple = "apple", "banana", "cherry"

>>> "apple" in mytuple
??

>>> "p" in "apple"
??
```

tuple operators +, +=, *, *=

```python

t1 = (1, 2, 3)
t2 = ("abc", "def")

# what do you think this prints?
print( t1 + t2 )

t1 += t2
t1 *= 2

# what do you think this prints?
print( t1 )



```

Add single item to tuple

```python

# create empty tuple
t2 = ()

# add two items to our empty tuple
t2 += "abc", "def"

# add one more item
t2 += ("ghi",)

# add one more item
moreText = "jkl"
t2 += (moreText,)


```

max, min, sum

```python

t = (10, 2, 3, 5)

# what do you think this prints?
print( len(t) )
print( sum(t) )
print( min(t) )
print( max(t) )
```

for-loop syntax

```
for [variable] in [sequence]:
    [ code block ]
```

- start with the word **for**
- then a space, followed by some *[variable]*
    - this *[variable]* will take on all values in the *[sequence]*, one at a time
- followed by a colon and a newline
- then add your <u>indented</u> *code block* (things to repeat for every value in the sequence)

for loop

```python

#what do you think is the output of this code?
tpl = (10, 2, 3, 5)

for x in tpl:
    print( x )
```

for loop

```python

#what do you think is the output of this code?
tpl = (10, 2, 3, 5)

for x in tpl:
    print( "hello" )
```

for loop

```python

#what do you think is the output of this code?

for ch in "hello":
    ch += "!"
    print( ch )
```

- for loop
  - break, continue
- range
- iterable
  - iter, next
  - convert iterable to tuple
- index, negative index
- slices
- enumerate

for loop - break and continue

```python

#what do you think is the output of this code?
for x in (10, 2, 3, 5):
    print( x )


#what do you think is the output of this code?
for x in (10, 2, 3, 5):
    if x == 2:
        break
    print( x )
```

for loop - break and continue

```python

#what do you think is the output of this code?
for x in (10, 2, 3, 5):
    print( x )


#what do you think is the output of this code?
for x in (10, 2, 3, 5):
    if x == 2:
        continue
    print( x )




```

for loop - break and continue

```python

#what do you think is the output of this code?
for x in (0, 1, 2, 3, 4, 5):
    print( x )

#what do you think is the output of this code?
for x in (0, 1, 2, 3, 4, 5):
    if x % 2 != 0:
        continue
    print( x )

# did we need to say "if x%2!=0:..."?
#   could we have just written "if x%2:..."?


```

for loop - break and continue

```python

#what do you think is the output of this code?
for x in (0, 1, 2, 3, 4, 5):
    print( x )

#what do you think is the output of this code?
for x in (0, 1, 2, 3, 4, 5):
    if x % 2:
        continue
    print( x )
```

range

```python

#instead of specifying every number from 0 to 5...
for x in (0, 1, 2, 3, 4, 5):
    print( x )


#we can just use range(6) - 6 is the stopping point
for x in range(6):
    print( x )
```

range

```python

#what do you think is the output of this code?

for x in range(100):
    print( "hello" )
```

range(*start*, *stop*)

```


# print all numbers from 1 to 10
for x in range(1, 11):
    print( x )


# what do you think this prints?
for x in range(5, 7):
    print( x )
```

range(*start*, *stop*, *step*)

```python


# what do you think this prints?
for x in range(10, 21, 2):
    print( x )


# what do you think this prints?
for x in range(0, 50, 10):
    print( x )
```

nested for loops

```python

#what do you think is the output of this code?
for i in range(1, 5):
    for j in range(1, 5):
        print(f"{i} {j}")
```

nested for loops

```python

#what do you think is the output of this code?
for i in range(1, 5):
    for j in range(i, 5):
        print(f"{i} {j}")
```

What exactly is range?

- The range(...) function returns a range object
    - str(...) return a string
    - int(...) return an int
    - tuple(...) return a tuple
    - range(...) return a range
- Just like a tuple or a string, a range object is an **iterable** object (i.e., a sequence-like object)
- An iterable is any Python object capable of returning its members **one at a time**, permitting it to be iterated over in a for-loop.

iter() and next()

```python
#what do you think is the output of this code?
r = range(1, 5)

rIter = iter(r)

print( next(rIter) )
print( next(rIter) )
print( next(rIter) )
print( next(rIter) )
print( next(rIter) ) # once there are no more next's, we get error
```

iter() and next()

```python
#what do you think is the output of this code?
r = "hello"

rIter = iter(r)

print( next(rIter) )
print( next(rIter) )
print( next(rIter) )
print( next(rIter) )
print( next(rIter) )
print( next(rIter) ) # once there are no more next's, we get error
```

next(*iterator*, *defaultValue*)

```python
#what do you think is the output of this code?
r = "hello"

rIter = iter(r)

print( next(rIter, None) )
print( next(rIter, None) )
print( next(rIter, None) )
print( next(rIter, None) )
print( next(rIter, None) )
print( next(rIter, None) )
```

convert any iterable to a tuple

```python

r = range(5)
s = "hello"

t = tuple(r)
t += tuple(s)

# what do you think this will output
print(t)
```

Ordered sequences

- tuples, strings, and range are all **ordered**
  - that means that the first item in these sequences is always first, second is always second, and so on
- any child of an ordered sequence can be accessed by its index (i.e., its position in the sequence)
  - first item is at index 0, second at index 1, etc.

```
>>> ("hi", "hello", "bye")[0]
"hi"
>>> ("hi", "hello", "bye")[1]
"hello"
```

Item at index

- syntax for accessing an item at a given index (i.e., position) in an ordered sequence:
    - *sequence*[*index*]
        - specify index in square brackets
        - index must be a whole number
            - index 0 is the very first item in the sequence
            - index 1 is the second item in the sequence
        - index can be negative
            - index -1 is the last item in the sequence
            - index -2 is second-to-last item in the sequence

item at index within a sequence

```python
items = 'socks', 'shoes', 'shirt', 'pants'

#what do you think this prints?
print( items[0] )

#what do you think this prints?
print( items[1] )

#what do you think this prints?
print( items[-1] )

#what do you think this prints?
for i in range(len(items)):
    print(f"item {i+1}: {items[i]}")

```

enumerate(*seq*)

- enumerate is another function that returns an iterable object
- specifically, the enumerate(*seq*) iterable will iterate over *seq*, and return a pairs of values for each iteration
    - the first value in each pair will be the index
    - the second value will be whatever the next value in *seq* was supposed to be

enumerate

```python

for i, char in enumerate("hello"):
    print(f"the character at index {i} is {char}")

#Expected output:
# the character at index 0 is h
# the character at index 1 is e
# the character at index 2 is l
# the character at index 3 is l
# the character at index 4 is o
```

enumerate

```python

#what do you think this code outputs?

items = 'socks', 'shoes', 'shirt', 'pants'

for i, item in enumerate(items):
    print(f"item {i+1} is {item}")
```

Subsequences and slices

- You can get just a part of any sequence by specifying a slice insides square brackets:
  - *sequence*[*indexStart* : *indexStop*]
    - gets all items from *indexStart*, up until, but not including *indexStop*
  - *sequence*[*indexStart* :]
    - gets all items from *indexStart* until the end of the sequence
  - *sequence*[: *indexStop*]
    - gets all items from up until, but not including *indexStop*

Slices

```python
items = 'socks', 'shoes', 'shirt', 'pants'

#what do you think this prints?
print( items[0:2] )

#what do you think this prints?
print( items[1:] )

#what do you think this prints?
print( items[:-1] )

#what do you think this prints?
print( "hello world"[-5:] )


```

Subsequences and slices

- `You can even specify a step-size:`
    - `*sequence*[ *indexStart* : *indexStop* : *step* ]`
    - `*sequence*[ *indexStart* :: *step* ]`
    - `*sequence*[ : *indexStop* : *step* ]`
    - `*sequence*[ :: *step* ]`

Slices with step-size

```python

items = 'socks', 'shoes', 'shirt', 'pants'

#what do you think this prints?
print( items[::2] )

#what do you think this prints?
print( items[1::2] )

#what do you think this prints?
print( "hello"[::-1] )
```

Assignment 6 - due before next lecture

- Build a text-based RPG game; below is the game loop:
  1. `print("\033[H\033[2J", end="") #clear screen`
  2. display current health, number of items, and coins
  3. display items (use for-loop to show items)
  4. encounter random thing
  5. if this thing is an item and player does not yet have it, ask them if they want it...
  6. else if this thing is a creature:
     - if it's a monster, it attacks...
     - ask player if they want to fight creature...
  7. `input("Press ENTER to keep walking")`
  8. repeat step 1

Assignment 6 - due before next lecture

5. if this thing is an item and player does not yet have it, ask them if they want it
    - if player response is not 'n' or 'N', take the item

- items are all pieces of armor:
    - shield, helmet, boots, chest-plate, gauntlets

Assignment 6 - due before next lecture

```
6.  if this thing is a creature:
      ■ if it's a monster, it attacks player
          ● attack amount = 7 - number of armor items
          ● let player know they were attacked by attack amount
          ● reduce player health by attack amount
              ○ if player health <= 0, game ends, player loses
      ■ ask player if they want to fight creature
          ● if player doesn't say 'n' or 'N'
              ○ they they kill creature
                  ■ increase player score
                      ● by 10 for a monster
                      ● by 1 for any other creature
                  ■ if score >= 100, game ends, player wins!
```

Assignment 6 - due before next lecture

- Here's how to start your rpg.py file:
  - `import random`
  - create a constant CREATURES which is a tuple ('monster', 'rabbit', 'fox', 'rat')
  - create constant ITEMS which is a tuple ('helmet', 'shield', 'boots', 'chest plate', 'gauntlets')
  - create constant OTHER_THINGS which is a tuple ('bush', 'big tree', 'rock')
  - create constant ALL_THINGS which is a combination of above three tuples
    - `#use random.choice(ALL_THINGS) to select random thing`
  - set initial player health to 20

Assignment 6 - extra credit 1 (10pts)

- create a copy of your rpg.py, call it rpgavg.py
- alter rpgavg.py to run the game 1000 times and report average score and win-rate, assuming user always says 'yes' to every action
    - remove all input statements, so that the code runs without need for user input
    - collect win/loss and scores
    - report win-rate, report average score

Assignment 6 - extra credit 2 (10 to 50pts)

- Alter the game to make it more interesting; e.g.,
    - make player choices more meaningful
    - add fight sequences
        - creatures don't die from single hit
        - player can run away in the middle of a fight
    - all creatures can fight back
    - each creature type has different health pool and attack values
    - add more random elements to the game
    - monsters might drop healing potions
    - different levels have different monsters

```python
# clear screen
print("\033[H\033[2J", end="")
```