Section 6: Sequences

CHAPTER 4 AND 5

General Description

- **"Normal" variables**

  x = 19

  - The name "x" is associated with a single value
- **Sequence variables**:

  T = (5, 17, 19, -1)

  - A name which is associated with a group of values.

Types of Python sequences

- Immutable (create once, can't edit)
  - **strings**: a collection of glyphs
  - **tuples**: a collection of python objects
  - **range objects**: a collection of integer (index #'s)
- Mutable (can add / remove / etc)
  - **lists**: like tuples, but slower
  - **dictionaries**: a mapping from key to value

Creating sequences with existing data

```
s = “abc”                                        # string
t = (1, 4.2, “def”)                              # tuple
L = [1, 4.2, “def”]                              # list
D = {“joe”: 1, “bob”: 4.2, “sue”: “def”}         # dictionary
```

- Key word: **delimiter**

Creating sequences with existing data

- As before

```
s = “abc”                                          # string
t = (1, 4.2, “def”)                          # tuple
L = [1, 4.2, “def”]                          # list
D = {“joe”: 1, “bob”: 4.2, “sue”: “def”}  # dictionary
```

- New operations

```
len(s)          # 3
len(t)          # 3
len(L)          # 3
len(D)          # 3
“a” in s                # True
4.2 in L                # True
“bob” in D              # True
4.2 in D                # False – see why?
s + “yyy”               # abcyyy
t + (9, 10)             # (1, 4.2, “def”, 9, 10) # no dictionaries!
```

Indexing Sequences (except Dictionaries)

- Index number identifies a position.

```
s = "Oranges"
```

-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6
O	r	a	n	g	e	s

- Examples:

```
>>> s[0]          # 'O'
>>> s[2]          # 'a'
>>> s[-1]         # 's'
>>> s[-2]         # 'e'
```

- The indexed value can be used anywhere a "normal" variable / constant would

Indexing Dictionaries

```
D = {“joe”: 1, “bob”: 4.2, “sue”: “def”}  # dictionary
print(D[“joe”])   # 1
print(D[“sue”])   # def
print(D[“xyz”])   # Error!
print(D)          # { “bob”: 4.2, “sue”: “def”, “joe”: 1}

Note: with dictionaries, there is no ordering!
```

Slicing Sequences

- Returns a section of a sequence
  - Doesn't work for Dictionaries.

```
>>> s = "Apples-Oranges"
```

-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	**p**	**p**	**l**	**e**	**s**	**-**	**O**	**r**	**a**	**n**	**g**	**e**	**s**

```
>>> s[1:4]      # "ppl"
>>> s[5:9]      # "s-Or"
>>> s[-6:-1]    # "range"
>>> s[-5:-7]    # ""
>>> s[10:]      # "nges"
>>> s[:6]       # "Apples"
>>> s[1:10:2]   # "plsOa"
>>> s[12:8:-1] # "egna"
>>> s[-1::-1]  # "segnarO-selppA"   (backward)
```

counting and indexing

```
L = [“ab”, “c”, “def”, “gh”]
L.count(“ab”)        # 1
L.count(“x”)         # 0
L.index(“c”)         # 1
L.index(“x”)         # Error!

# Note: none of these work on dictionaries.
```

list-only methods

```
L = [“a”, “b”, “c”]
L.append(“d”)
print(L)              # [a, b, c, d]
L.insert(1, “x”)
print(L)              # [a, x, b, c, d]
L.remove(“a”)         # The first one…
print(L)              # [x, b, c, d]
del L[-1]
print(L)              # [x, b, c]
L[0] = “y”
print(L)              # [y, b, c]
```

Micro-labs

- Frist one:
  - Have the user enter a string.
  - Convert it to lower-case (s = s.lower())
  - Determine if it's a palindrome
- Second one:
  - Find and print all prime numbers from 2 to n
  - Make a list of all numbers
  - Remove a number if it's evenly divisible by anything that comes before.

a new random function

- random.choice(sequence)
- Returns a random element of sequence.

```
s = "ABCDEFG"
print(random.choice(s))
```

- Good quiz question: do it manually.

for loops

- A type of repetition, but tied to a sequence.
- Syntax:

```
for var in seq:
        # for-loop block
```

- The number of items in seq controls the number of iterations.
- var is a variable, created / updated by python
  - A reference to the "current" item in the sequence.
- Equivalent to:

```
i = 0
while i < len(seq):
    var = seq[i]
    i += 1
    # for-loop block
```

for loop example

```
s = "ABCDE"
for c in s:
    print(c)
```

Outputs:

```
A
B
C
D
E
```

```
# Doing the same thing with a
while loop
i = 0
while i < len(s):
        c = s[i]
        print(c)
        i += 1
```

range objects

`range([start,] end [, step])`

- Treated as a sequence (sort of)
- Useful when used with for loops
- Example:

```
s = "ABCDE"
for i in range(len(s)):   # prints 0,1,2,3,4
    print(i)
for i in range(2,10,3):   # prints 2,5,8
    print(i)
```

Pulling back the pygame curtain

- Discuss how this “magic” used sequences:
  - Accessing RGB of color tuples
  - The key and mouse pressed (and pos) tuples
  - key_pressed lists
  - rectangles
  - …

2D Tuples / Lists

- A common name for a tuple with tuple (sub-)elements
- Example:

```
T = ((15,8,7), (3,2,9), (1,-2,10))
T[0]                        # (15,8,7)
T[0][1]                     # 8
T[0:2]                      # ((15,8,7), (3,2,9))
T[0][1:]                    # (8,7)
T[1] + T[2]                 # (3,2,9,1,-2,10)
T[1] + T[1]                 # (3,2,9,3,2,9)
T[1] * 3                    # (3,2,9,3,2,9,3,2,9)
T[1][1] + T[2][-1]          # 12
```

Examples!

- [scrolling star field]
- [simple particle-effects]
- [simple tile-worlds]