Dictionary

Only certain hashable types can be stored as a key in a dictionary, such as strings, numbers, tuples

```
dictionary = {
  "key": value
}
```

DefaultDict

Returns a default value if a key is not found

```
from collections import defaultdict

# Will return an empty list, if no key is found
dictionary = defaultdict(list)

dictionaery["items"] # []
```

Array

Implemented as a dynamic array under the scenes

```
arr = ["one", "two", 3]
arr[0] # "one"

arr.append("four")
arr[-1] # "four"
```

Tuple

Immutable arrays where all items must be defined at instantiation

```
arr = ("one", "two", 3)
arr[0] # "one"
```

String

Immutable array of characters, that can be iterated over

```
arr = "abcd"
arr[0] # "a"

# Unpacking
list(arr) # ["a", "b", "c", "d"]
# Repacking
"".join(list(arr))
```

Set

Mutable and allows for dynamic insertion and deletion of elements while maintain that each element is unique

```
set("apple") # {'a', 'p', 'l', 'e'}
set(["a", "p", "p", "l", "e"]) # {'p', 'a', 'l', 'e'}

# Unpacking
list(arr) # ["a", "b", "c", "d"]
# Repacking
"".join(list(arr))
```

Stack

A collection that supports last in, first out method, where like stacking a plate, the last one you put on is the first item in the stack

```
# Using a list, access and removal is O(1)
s = []
s.append("eat")
s.append("sleep")
s.append("code")

s.pop() # 'code'
s.pop() # 'sleep'
```

```
# Using a deque, access and removal is O(1)
from collections import deque
s = deque()

s.append("eat")
s.append("sleep")
s.append("code")

s.pop() # 'code'
s.pop() # 'sleep'
```

Queue

A collection that supports first in, first out method, where like if you were in a line, the first one into the line is the first one to leave the line

```
# Using a list, access and removal is O(n) bad
q = []
q.append("eat")
q.append("sleep")
q.append("code")

q.pop(0) # 'code' O(n)
q.pop(0) # 'sleep' O(n)
```

```
# Using a deque, access and removal is O(1)
from collections import deque
q = deque()
q.append("eat")
q.append("sleep")
q.append("code")

q.popleft() # 'code' O(1)
q.popleft() # 'sleep' O(1)
```

Priority Queues

A collection that orders information in a way that gives you access to the smallest or the largest comparable value in it's set

```
# Using a list, and a tuple to manually determine importance
q = []
# Insertion is O(n)
q.append((2, "code"))
q.append((1, "eat"))
q.append((3, "sleep"))
# Remember to re-sort every time a new element is inserted,
# or use bisect.insort(), this leads to O(n log n)
q.sort(reverse=True) #

q.pop # (1, "eat")
```

```
# Using a heapq, access (heappush) and removal (heappo) is O(log n)
import heapq
q = []
heapq.heappush(q, (2, "code"))
heapq.heappush(q, (1, "eat"))
heapq.heappush(q, (3, "sleep"))

heapq.heappop(q) # (1, 'eat')
```

Heapify

Turns a list into a heap

```
import heapq

H = [21,1,45,78,3,5]

# Convert to a heap O(n)
heapq.heapify(H) # [1, 3, 5, 78, 21, 45]

# Add element
heapq.heappush(H,8) # [1, 3, 5, 78, 21, 45, 8]
```

Heap replace

Removes the smallest element and then bubbles down the value

```
import heapq

H = [21,1,45,78,3,5]
# Create the heap
heapq.heapify(H) # [1, 3, 5, 78, 21, 45]

# Replace an element O(log n)
heapq.heapreplace(H,6) # [3, 6, 5, 78, 21, 45]
```