Heuristics

Lab Section 2

Today's Lab

- We will explore these exercises:
- name.py
- convert.py -- currency
- phrase.py (string analysis)
- cat.py
- loops.py
- wrongLoops.py
- height.py
- ten.py – in class assignment

converter.py

- Convert the given input of US $ money (integer) to euros.
- # This program converts from US $ to Euro

- us_money = int(input ("Aaditya asks Money value in US $ "))

- euros = us_money / 1.16
- print("US$", us_money, "= Euro ", euros)

height.py

- Get the user, gender and height information, and tell them if they are taller/shorter/equal to the average.
- The respective average heights for males and females are:
- Males: 175 cm, Females: 165 cm
- How many inputs do we need?
- You can only use strings.
- Checkout the (height.py) script.

name.py

- Get user name, make a string asking user for money and then print the resultant string in reverse.
  - #
  - print("hello, from NYU")
  - x = input ("Dude, what is your name? ")
  - print("Hello, " , x , "! How are you, " , x , "?")
  - print("Hey could you lend me some money " + x +"?")
  - print(("Hey could you lend me some money " + x +"?")[::-1])
- What is “[::-1]”? String slicing to get a particular part of a string in a defined way. Let’s discuss string more along with lists and functions.

Strings

- Strings are one of the data types in Python
- Strings are sequences of characters
- Strings are defined by single or double quotations (‘ ’, “ ”)
- Example:
  >>> name = ‘python’

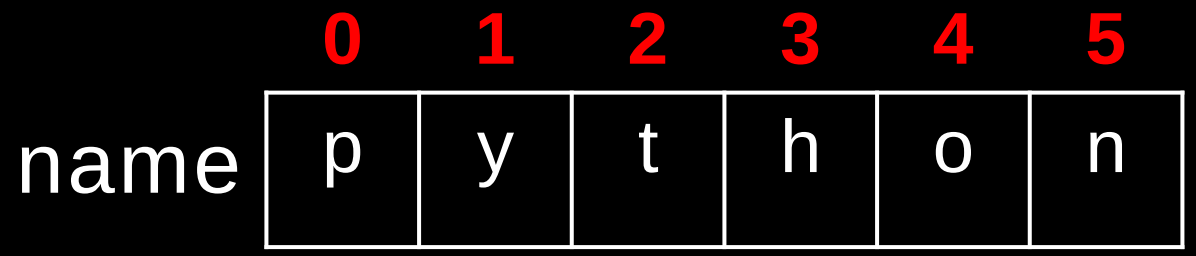
String **Indexing**

- You can access individual characters by using square brackets after the string:

- >>> character = string[index]

- Arguments:
  - index is an integer representing the index

String Indexing

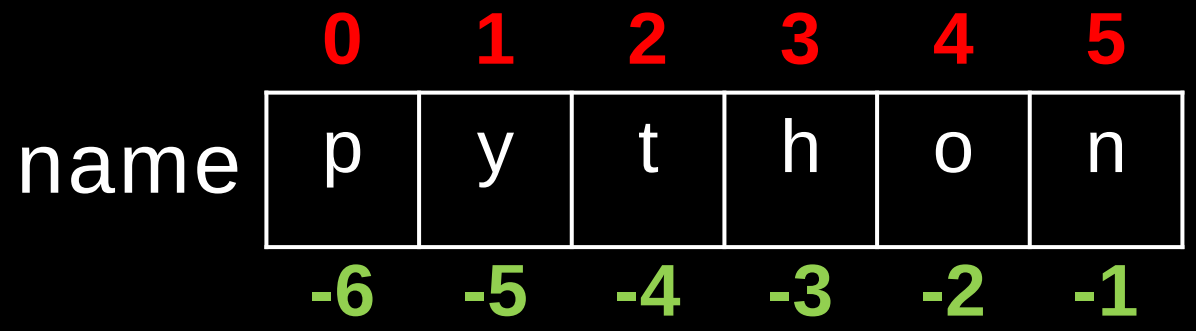


- What do you think this will give us?

```
>>> name[1]
y
```

- Index is an offset from the beginning of the string
  - So, the offset to the first letter is 0

String Indexing



- What about the following:

```
>>> name[1.5]
>>> name[6]
>>> name[-1]
```

TypeError

IndexError

String Slicing (Substrings)

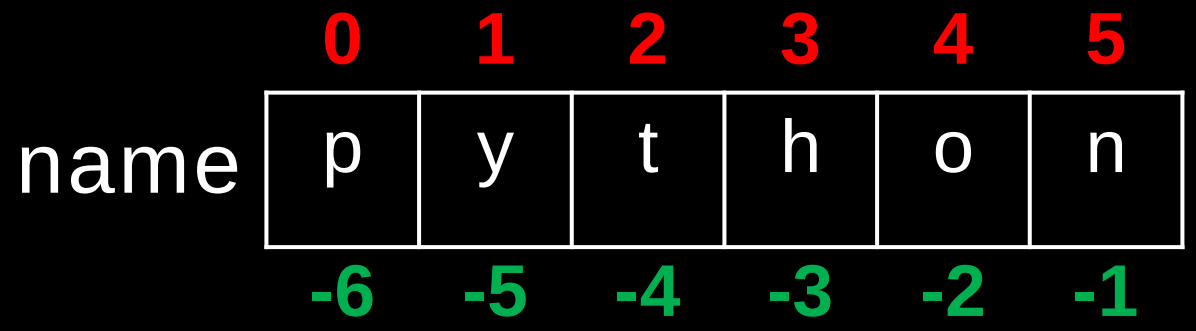
- To get a slice of a string, use the following expression:
  >>> substring = string[start:end:step]

- Arguments:
  - start: first character (inclusive) in the slice
  - end: last character (exclusive) in the slice (optional)
  - step: step size (optional)
    - default value +1

- It returns the substring according to the arguments

- start, end and step must be integers and can be positive or negative
- |start| or |end| can be larger than the string length

String Slicing Example




```
>>> name[1]
>>> name[-5]
>>> name[0:2]
>>> name[-6:-1]
>>> name[-6:]
>>> name[-1:-6]
>>> name[0:5:2]
>>> name[-1:-7:-1]
>>> name[::-1]
```

Strings are Immutable

- Since strings are immutable (from link given last time), you can not change or re-assign characters within strings

```
>>> name = 'python'
>>> name[0] = 'b'        Type Error
```

- However, you can create a new string

```
>>> new_name = 'b' + name[1:]
>>> print(new_name) bython
```

String Length

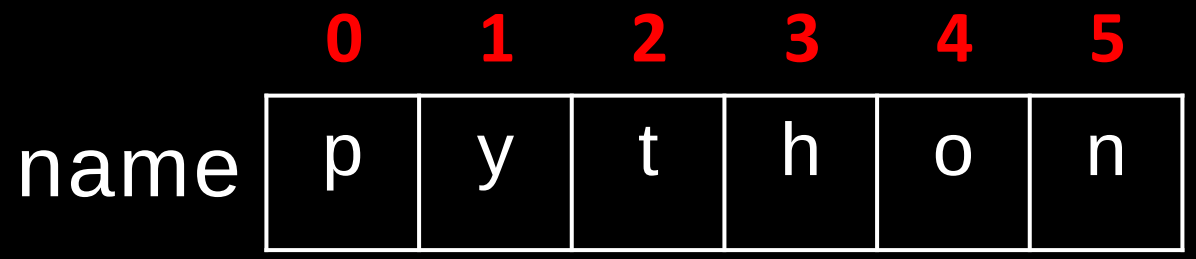
- Python has a built-in function that can be used to get the string length

- len(string) returns the total count of characters in the string

```
>>> name = ‘python’
>>> length = len(name)
>>> print(length)
6
>>> print(name[length])
```

IndexError

String Length



- There are 6 characters in name, but index [6] does not exists!

```
>>> name = ‘python’
>>> length = len(name)
>>> print(name[length-1])
n
```

Strings Comparison

- You can compare strings by using the equality operator ==
- It compares the numeric value of the strings
- It returns either True or False

```
>>> print('apple' == 'banana')
False
>>> print('apple' == 'apple')
True
```

Strings Comparison

- Remember: upper and lower case are not the same

```
>>> print(‘Apple’ == ‘apple’)
False
```

Strings Comparison

- You can also check if strings are not equal using != operator

```
>>> print('Apple' != 'apple')
True
```

The in Operator

- The **in** operator is a operator that checks existence of a string in another string:

```
>>> print('p' in 'apple')
True
>>> print('ple' in 'apple')
True
```

String Methods

String Methods

- Python has a number of useful string methods that can help manipulate strings:
  - Change letters/sentences' case
  - Replace characters
  - Search for characters
  - Count characters
- Link to explore the methods:

String Methods

- Since strings are objects in Python, they use the dot (.) notation

object.method(argument)

- All string methods return a value! They do not change the string!

- Example:

```
>>> name = “python”
>>> upper_name = name.upper()
>>> print(upper_name)
PYTHON
```

String Methods

- Some string methods
  - .upper()
  - .lower()
  - .capitalize()
  - .title()

- All string methods return a value (string)! They do not change the string variable!

Replacing Substrings

- Replacing a substring in a string:
  >>> substring = string.replace(old, new, max)

- Arguments:
  - old: substring to be replaced
  - new: new substring to replace the old substring
  - max: number of replacements (optional)

- It returns a copy of the string in which the occurrences of old have been replaced with new

Example: Replacing Substrings

```
>>> name = ‘This is not an advertisement!’

>>> name.replace(‘is’, ‘was’)
>>> print(name)
‘This is not an advertisement!’

>>> new_name = name.replace(‘is’, ‘was’)
>>> print(new_name)
‘Thwas was not an advertwasement!’
```

How to replace only the word ‘is’?

```
>>> new_name = name.replace(‘ is ’, ’ was ’)
>>> print(new_name)
‘This was not an advertisement!’
```

Finding Substrings

- Finding a substring in a string:
  >>> index = string.find(sub, start, end)

- Arguments:
  - sub: substring to find
  - start: start index (inclusive) for the search (optional)
  - end: end index (exclusive) for the search (optional)

- It returns:
  - the lowest (positive) index of the first character where sub is found within the string
  - -1 if no match was found

Example: Finding Substrings

```
>>> name = 'python'
>>> print(name.find('p'))
0
>>> print(name.find('th'))
2
>>> print(name.find('th',3))
-1
>>> print(name.find('th',0,3))
-1
```

How about: `print(name.find('th',-4,-1))`

```
2
```

Counting String Occurrences

- Counting how many times a substring is present in a string:

  occurrences = string.count(sub, start, end)

- Arguments:
  - sub: substring to count
  - start: start index (inclusive) for the count (optional)
  - end: end index (exclusive) for the count (optional)

- It returns the number of occurrences of sub in the string

Example: Counting String Occurrences

```
>>> name = 'banana'
>>> print(name.count('a'))
3
>>> print(name.count('p'))
0
>>> print(name.count('na'))
2
>>> print(name.count('a', 3))
2
>>> print(name.count('a', 0, 3))
1
```

Sequences and Lists

Sequences

- A sequence is an object that holds multiple items of data
  - it stores the data one after the other

- In Python, there are several types of sequences
  - String: sequence of characters
  - Lists: sequence of items of any data type

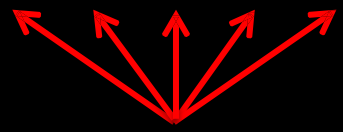
Lists

- Lists are used to store multiple items in a single variable
- Lists can contain
  - items from the same data type
  - items from different data types
- Lists are mutable (changeable)
  - List items can be modified after they have been created
  - List items can be added or removed from the list during runtime

Lists

- Lists are created using square brackets [ ]

```
>>> empty_list = [ ]
>>> odd_numbers = [1, 3, 5, 7, 9]
```



List items are (visually)
separated by commas (,)

- Lists allow duplicate values
- List items are ordered*

*ordered != sorted

Examples: Lists

```
>>> odd_numbers = [1, 3, 5, 7, 9]
```


```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’, ‘Robb’]
```


```
>>> mixed_list = [‘Jon’, 1, 2.5, ‘a’, True]
```

Overloaded Operators on Lists

- Similar to strings you can:
  - concatenate lists using the + operator
  - apply repetition using the * operator

- Examples:

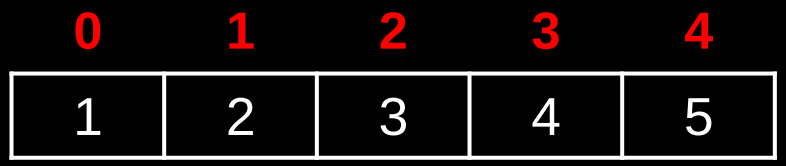
```
>>> list1 = [1, 2]
>>> list2 = [3, 4]
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4]
```

```
>>> list1 = [1, 2]
>>> num = 3
>>> list2 = list1 * num
>>> print(list2)
[1, 2, 1, 2, 1, 2]
```

List Indexing and Slicing

- List items are indexed

```
>>> numbers = [1, 2, 3, 4, 5]
>>> print(numbers[1])
2
```



- Slicing works too!

```
>>> print(numbers[2:4])
[3, 4]
>>> print(numbers[1:2])
[2]
```

- Note: Slicing always returns a List!

Strings vs. Lists

Strings

- name = ‘python’

- Strings are defined using single or double quotation
- Indexing/slicing
  - name[0] → ‘p’
  - name[2:4] → ‘th’

Lists

- numbers = [1, 2, 3, 4, 5]

- Lists are defined by items inside [ ]; items are separated by commas
- Indexing/slicing:
  - numbers[0] → 1
  - numbers[2:4] → [3, 4]

Strings vs. Lists
or
Immutability vs. Mutability

- Strings are immutable:

```
>>> name = 'python'
>>> name[0] = 'b'
```

Type Error

- Lists are mutable:

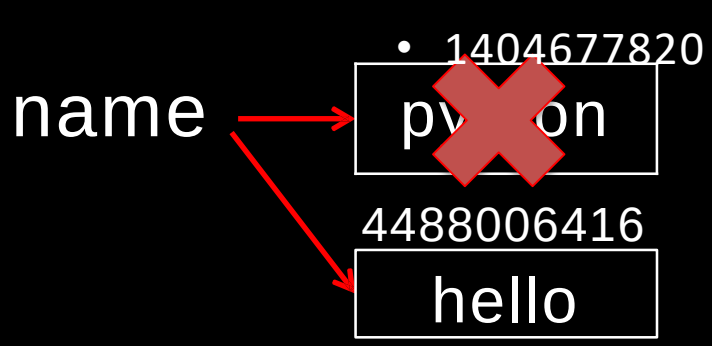
```
>>> even_numbers = [1, 4, 6, 8]
>>> even_numbers[0] = 2
>>> print(even_numbers)
[2, 4, 6, 8]
```

Immutability vs. Mutability

- Strings:
  - >>> name = ‘python’
  - >>> print(id(name))
    1404677820
  - >>> name = ‘hello’
  - >>> print(id(name))
    4488006416

- Lists:

```
>>> numbers = [1, 4, 6, 8]
>>> print(id(numbers))
7750390128
>>> numbers[0] = 2
>>> print(id(numbers))
7750390128
```



Copying Strings

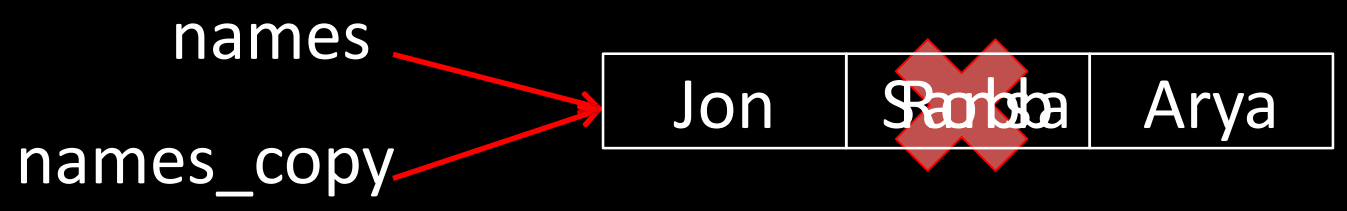
```
>>> name  = 'python'
>>> new_name  = name
>>> name = 'hello'

>>> print(name)
'hello'
>>> print(new_name)
'python'
```

Copying Lists

- Since Lists are mutable, list variables only store a reference to the object
- If you want to copy a list, you need to copy its items
- Just assigning the list to a new variable will **not** create a copy of the list
  - Both variables are pointing to the same memory location where the list is stored

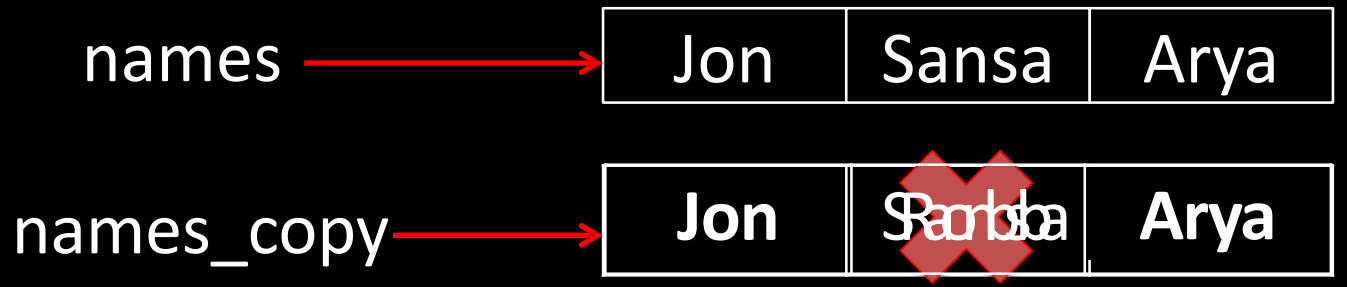
Example: Copying Lists

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’]
>>> names_copy = names
>>> names_copy[1] = ‘Robb’
>>> print(names)
[‘Jon’, ‘Robb’, ‘Arya’]
```



How to copy Lists then?

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’]
>>> names_copy = [] + names
>>> names_copy[1] = ‘Robb’
>>> print(names)
[‘Jon’, ‘Sansa’, ‘Arya’]
```



Lists Length

- You can check how many items are in a list using the len() function:

```
>>> names = [‘Jon’, ‘Sansa’, ‘Robb’]
>>> len(names)
3

>>> names = []
>>> len(names)
0
```

List Methods

Index method

- The index(item) method returns the index of the first element in the list that matches the item in the argument
  - An error is given if the item is not found in the list

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’]
>>> print(names.index(‘Sansa’))
1

>>> print(names.index(‘Cersei’))
>>> ValueError: ‘Cersei’ is not in list
```

The in Operator with Lists

- Remember the in operator for strings?
  - Checks if a substring exists in another string
  - Use the in operator with lists to check if an item is present in the list or not

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’]
>>> ‘Jon’ in names
>>> True

>>> ‘Cercei’ in names
>>> False
```

Adding Items to Lists

- The append(item) method is used to dynamically add an item to a list during runtime
- The item appended is added to the end of the list
- The append(item) method modifies the list
  - it does NOT return a new list

- Example:

```
>>> numbers = [1, 2, 3]
>>> numbers.append(4)
>>> print(numbers)
[1, 2, 3, 4]
```

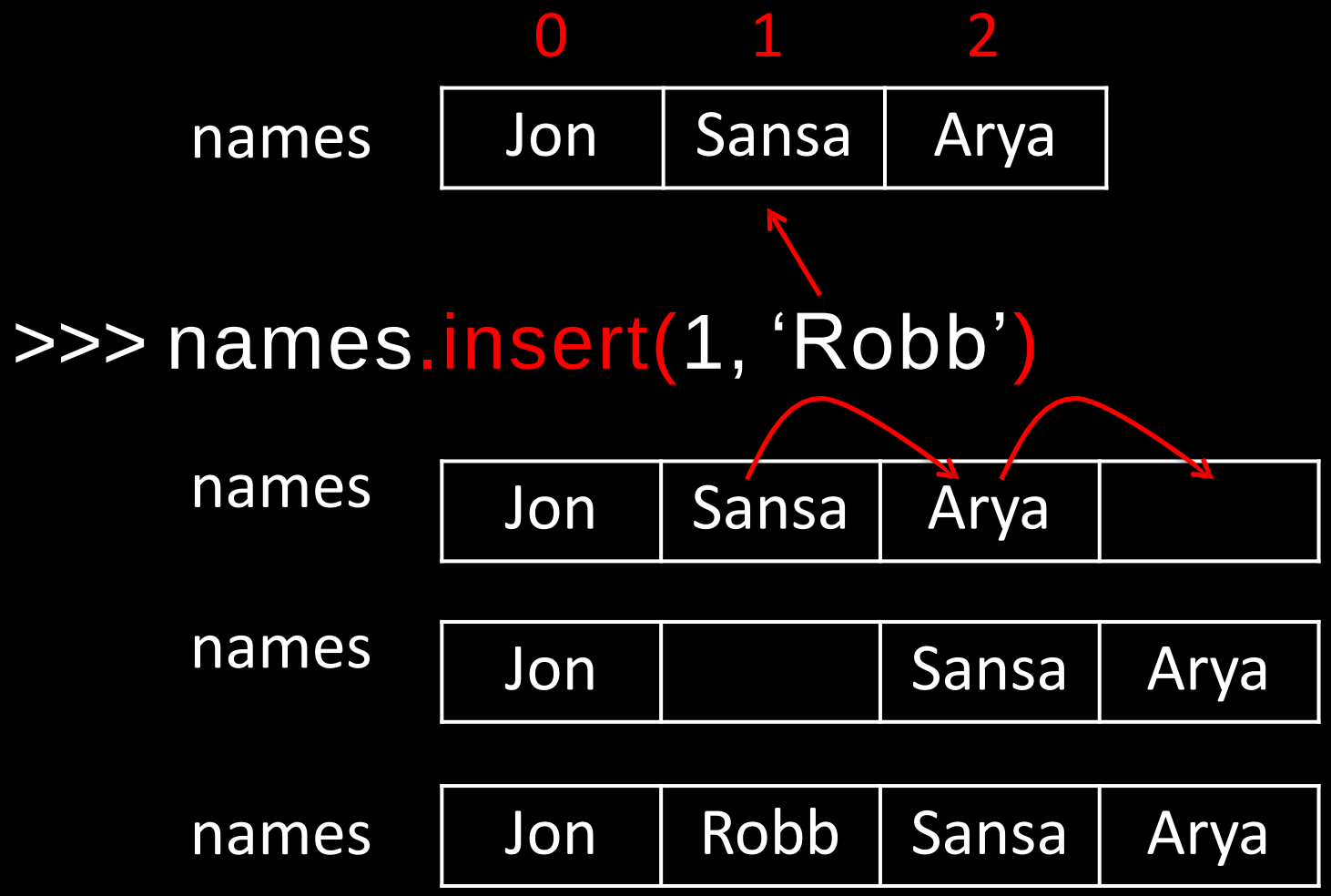
Never do:

```
>>> numbers = numbers.append(4)
>>> print(numbers)
None
```

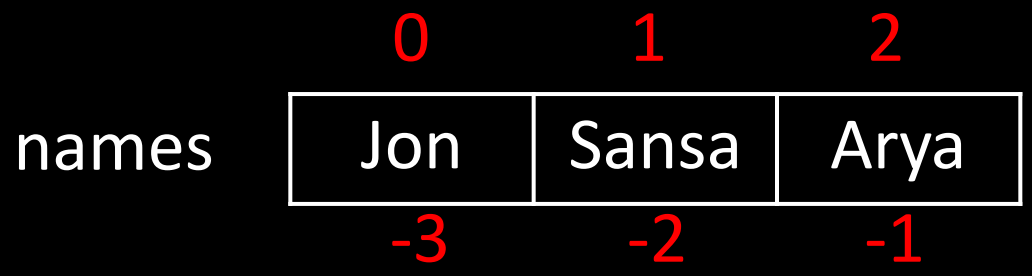
Inserting Items Into Lists

- The insert(index, item) method
  - inserts an item to the list at a specific index
  - does NOT return the new list

Example: Inserting Items



Example: Inserting Items



- What will happen if you use an invalid index?

```
>>> names.insert(50, 'Robb')
```

- What will happen if you use a negative index?

```
>>> names.insert(-3, 'Robb')
```

Other Useful List Methods and Functions

Method	Description
.sort()	Sort the items within the list in ascending order (from lower value to upper value)
.reverse()	Reverse the order of the items in the list
.count(item)	Counts how may times an item appears in the list
min (myList)	returns the element with the minimum value in the list
max (myList)	returns the element with the maximum value in the list

You can find more list methods here: https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

Convert Strings to Lists

- Remember type casting?
  int(), float() and str()?

- A string can be changed into a list using the list() type cast:

```
>>> name = 'python'
>>> my_list = list(name)
>>> print(my_list)
['p', 'y', 't', 'h', 'o', 'n']
```

Converting Lists to Strings

- How about a string of words?

```
>>> sentence = 'I love python'
>>> print(list(sentence))
['I', ' ', 'l', 'o', 'v', 'e', ' ', 'p', 'y', 't', 'h', 'o', 'n']
```

Splitting Strings

- The split(separator) method splits a string

string.split(separator)

- The separator argument is optional; by default " "
- It returns a list of strings

```
>>> sentence = 'I love python'
>>> print(sentence.split())
    ['I', 'love', 'python']
```

Splitting Strings

- You can also define the separator

```
>>> sentence = 'I-love-python'
>>> sentence.split('-')    ['I', 'love', 'python']
>>> sentence.split('o')    ['I-l', 've-pyth', 'n']
>>> sentence.split('love')    ['I-', '-python']
>>> sentence.split()          ['I-love-python']
```

Lists to Strings

- The join(list) method does the opposite of the split() method
- It joins all items in the list into one string

string.join(list)

- It returns a string by joining all list elements, separated by the string

- Note: list must contain string items!

Example: Lists to Strings

```
>>> separator = “ “
>>> word_list = [‘I’, ‘love’, ‘python’]
>>> joined_string = separator.join(word_list)
>>> print(joined_string)
I love python

>>> separator = “_”
>>> joined_string = separator.join(word_list)
>>> print(joined_string)
I_love_python
```

Removing an Item From a List

- If you want to remove an item from a list, there are three different ways to do this:
  - list.remove(item) removes the first occurrence of the item within the list (item is NOT returned!)
  - del list[index] removes the item at the specific index from the list
  - list.pop() removes the last item from the list and returns it
    - list.pop(index) removes an item at the specific index from the list and returns it

Example: remove() Method

- 0 1 2 3
- names

- >>> names.remove('Sansa')

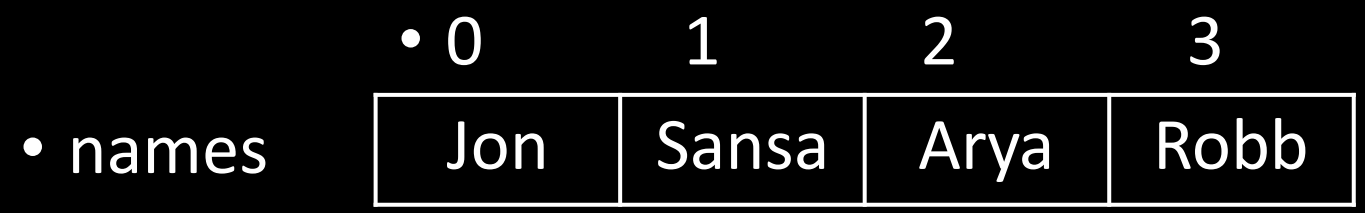
names

names

Never do:

```
>>> names = names.remove("Sansa")
>>> print(names)
None
```

Example: del Statement



- >>> del names[2]

>>> del names[5] IndexError

Example: pop() Method


```
>>> name = names.pop()
>>> print(name)
Robb
```

phrase.py (string analysis)

- Get a phrase from the user and analyze it by printing its:
- 1) length,
- 2) first letter, 3) last letter, 4) middle letter,
- 5) print it backward,
- 6) print it in upper case, 7) print it in lower case,
- 8) print its title, 9) split it on 'e’,
- 10) count words in it.
- Check out the (phrase.py) script.

Revision Exercise – Loop while

- Keep on getting the user-preferred coding language until they say yes to python being their favorite.
- At any input, if the given answer is longer than ten characters, do tell them “it’s a long answer”.
- Check out the (loops.py) script.

Revision Exercise – Nested Loop

- Given as homework to explore the nested loops.
- What are they, and why do we need them?
- nested loop is a loop that is contained within another loop.
- Nested loops can be useful when you want to perform an operation on every element in a multi-dimensional data structure, such as a two-dimensional array. For example, you might use a nested loop to iterate over the rows and columns of a matrix and perform some operation on each element.
  - Create a 2D chess board.
- Or simply
- Nested loops can also be used to perform an operation on all possible combinations of elements from two or more data sets. For example, you might use a nested loop to compare every element in one list with every element in another list and perform some operation on the elements that meet certain criteria.
  - * to pick a match between teams that never played each other.

Nested Loops

- Nested for loops are a loop inside a loop

Example:

```
message_list = [“Let’s”, “try”, “all”, “combinations!”]
for n in range(10):
  for item in message_list:
    print(item, end= “ “)
for word1 in message_list:
  for word2 in message_list:
    print(item, end= “ “)
```

Nested Loops

- Example:

```
for outer_num in range(2):
    for inner_num in range(3):
        print(str(outer_num) + str(inner_num))
```

- Take pen an paper and think what the output of the code is!

Revision Exercise – Debug

- Check out the (wrongloops.py) script and identify issues with each of the nested loops.

Final Exercise – In class assignment

- Do the game of 10 questions for a number between 1 and 1000.
- Is the number equal, greater than, or less than x?
- People have to answer something like "equal, less, greater"
- Initially, the lower bound is 1 and the upper bound is 1000.
- Each time you get an answer, you either raise the lower bound
- or lower the upper bound.
- If the respondent is inconsistent, tell them that they are lying.
- But be nice about it.

End