Exposure CS 2021 for CS1

Chapter 13 Section 1-2 Slides

Introduction to Data Structures and Array Syntax

PowerPoint Presentation
created by:
Mr. John L. M. Schram
and Mr. Leon Schram
Authors of Exposure
Computer Science

Section 13.1

Introduction to Data Structures

Simple Data Type Definition

A *simple data type* is a data type that can only store one value.

Examples of *simple data types* are *integers*, *real numbers* and *Boolean* values.

Data Structure Definition

A *data structure* is a data type that can store more than one value.

Examples of *data structures* are *arrays*, *records* and *files*.

Array Definition

An *array* is a data structure with one or more elements of the same type.

Every element of the array can be accessed directly.

A one-dimensional array is frequently also called a *vector*.
A two-dimensional array is frequently also called a *matrix*.

The *array* is the first historical data structure which was introduced in the language *FORTRAN*.

1D Array/Vector Examples

Index	0	1	2	3	4	5	6	7
Value	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**

0	1	2	3	4	5	6	7	8
1.1	**2.2**	**3.3**	**4.4**	**5.5**	**6.6**	**7.7**	**8.8**	**9.9**

0	1	2	3	4	5	6	7	8	9	10	11	11	13	14	15	16	17	18	19	20	21	22	23	24	25
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

0	1	2	3	4	5
John	**Greg**	**Maria**	**Heidi**	**Diana**	**David**

0	1	2	3	4	5
True	**False**	**True**	**False**	**True**	**False**

2D Array/Matrix Examples

Row/Col Indexes	0	1	2
0	**100**	**150**	**200**
1	**250**	**300**	**350**
2	**400**	**450**	**500**
3	**550**	**600**	**650**
4	**700**	**750**	**800**
5	**850**	**900**	**950**

Row/Col Indexes	0	1	2	3
0	**1.0**	**3.2**	**4.9**	**5.7**
1	**2.4**	**9.8**	**3.5**	**7.6**
2	**5.6**	**6.2**	**8.7**	**2.1**
3	**8.1**	**6.5**	**2.3**	**3.2**

Row/Col Indexes	0	1	2	3	4	5	6	7	8	9	10	11	12
0	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
1	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

Record Definition

A *record* is a data structure with one or more elements, called *fields*, of the same or different data types.

The language *FORTRAN* did not have *records* which is why it was NOT good for business. *COBOL* (Common Business Oriented Language) introduced the *record* data structure.

Record Examples

Student Record

Field	Value
firstName	John
midInitial	Q
lastName	Public
address	811 Fleming Trail
grade	10
gpa	3.88
classRank	57
honorRoll	True
serviceHours	29

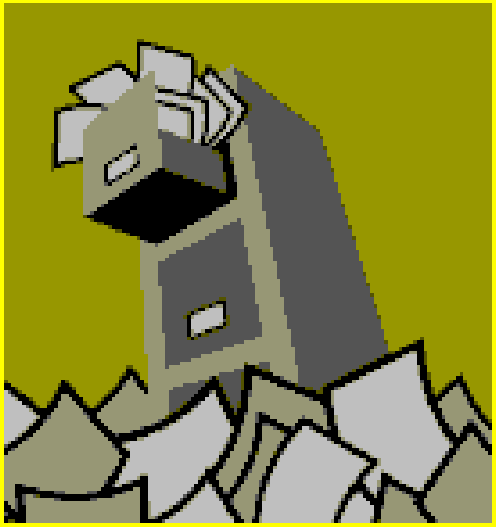
Employee Record

Field	Value
firstName	John
midInitial	Q
lastName	Public
address	811 Fleming Trail
salary	98765.43
position	CEO of Operations
officeNumber	2313
yearsWithUs	12
vested	False

File Definition

A *file* is an *internal* data structure – with an unspecified number of elements of the same type – assigned to an *external* file name.

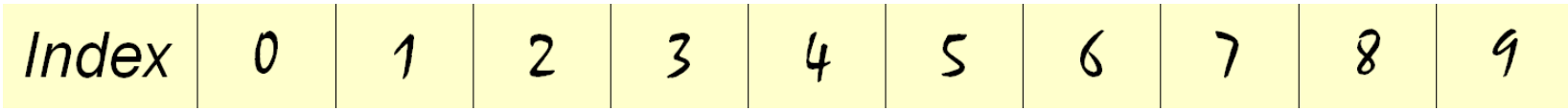
The file data structure allows transfer of data between internal and external storage.



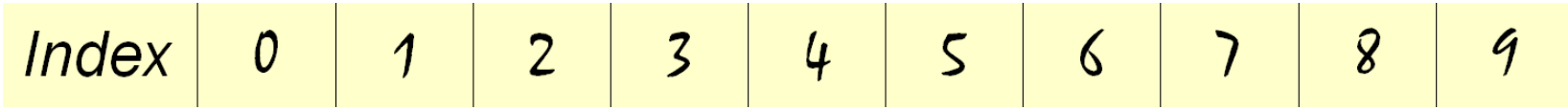
Section 13.2

Array Syntax

```
1  # ArraySyntax01.py
2  # This program demonstrates the creation of an
3  # array of integers in Python and demonstrates
4  # how to access individual array elements.
5
6
7  list = [100,101,102,103,104,105,106,107,108,109]
8
9  print()
10 print(list[0])
11 print(list[1])
12 print(list[2])
13 print(list[3])
14 print(list[4])
15 print(list[5])
16 print(list[6])
17 print(list[7])
18 print(list[8])
19 print(list[9])
```



```
1  # ArraySyntax01.py
2  # This program demonstrates the creation of an
3  # array of integers in Python and demonstrates
4  # how to access individual array elements.
5
6
7  list = [100,101,102,103,104,105,106,107,108,109]
8
9  print()
10 print(list[0])
11 print(list[1])
12 print(list[2])
13 print(list[3])
14 print(list[4])
15 print(list[5])
16 print(list[6])
17 print(list[7])
18 print(list[8])
19 print(list[9])
```




```
----jGRASP exec: python ArraySyntax01


100
101
102
103
104
105
106
107
108
109


----jGRASP: operation complete.
```

Array Index Note

Python arrays indicate individual elements with an index inside [brackets] following the array identifier, like **list[3]**

The array index is always an integer.

The index of the first item in the array is **0**.

In an array of **N** elements, the index of the last item is **N-1**.

```
1  # ArraySyntax02.py
2  # This program demonstrates a more efficient way
3  # to "traverse" an array by using a <for> loop.
4
5
6  list = [100,101,102,103,104,105,106,107,108,109]
7
8  print()
9  for k in range(10):
10     print(list[k])
11
```

Index	0	1	2	3	4	5	6	7	8	9

```
----jGRASP exec: python

100
101
102
103
104
105
106
107
108
109

----jGRASP: operation c
```

```
 1 # ArraySyntax03.py
 2 # This program demonstrates an array of characters/strings,
 3 # an array of real numbers and an array of Boolean values.
 4
 5
 6 letters = ['A','B','C','D','E','F','G','H','I','J','K','L','M',
 7             'N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
 8 print()
 9 for k in range(26):
10    print(letters[k],end=" ")
11 print()
12
13 gpas = [3.9,2.75,2.1,1.65,4.0,3.25,2.45,0.95,3.88,3.75]
14 print()
15 for k in range(10):
16    print(gpas[k],end=" ")
17 print()
18
19 booleans = [True,False,True,False,True]
20 print()
21 for k in range(5):
22    print(booleans[k],end=" ")
23 print()
```

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

3.9 2.75 2.1 1.65 4.0 3.25 2.45 0.95 3.88 3.75

True False True False True
```

```
 6 letters = ['A','B','C','D','E','F','G','H','I','J','K','L','M',
 7              'N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
 8 print()
 9 for k in range(26):
10     print(letters[k],end=" ")
11 print()
12
13 gpas = [3.9,2.75,2.1,1.65,4.0,3.25,2.45,0.95,3.88,3.75]
14 print()
15 for k in range(10):
16     print(gpas[k],end=" ")
17 print()
18
19 booleans = [True,False,True,False,True]
20 print()
21 for k in range(5):
22     print(booleans[k],end=" ")
23 print()
```

```
1 # ArraySyntax04.py
2 # This program demonstrates another string array.
3 # This time names are stored.
4 # NOTE: See what happens when you add or
5 #       remove names from the <names> array.
6
7
8 names = ["John","Greg","Maria","Heidi","Diana","David"]
9 print()
10 for k in range(6):
11     print(names[k],end=" ")
12 print()
13
```

```
  ----jGRASP exec: python ArraySyntax04

John Greg Maria Heidi Diana David

  ----jGRASP: operation complete.
```

```
 6
 7
 8 names = ["John","Greg","Maria","Heidi","Diana","David"]
 9 print()
10 for k in range(6):
11     print(names[k],end=" ")
12 print()
13
```

```
  ----jGRASP exec: python ArraySyntax04

John Greg Maria Heidi Diana David

  ----jGRASP: operation complete.
```

```
 6
 7
 8 names = ["John","Greg","Maria","Heidi","Diana","David"]
 9 print()
10 for k in range(6):
11     print(names[k],end=" ")
12 print()
13
```

Try This!

Add one or more names to this list. Will they show up in the output? Why?

```
----jGRASP exec: python ArraySyntax04

John Greg Maria Heidi Diana David

----jGRASP: operation complete.
```

```
 6
 7
 8 names = ["John","Greg","Maria","Heidi","Diana","David"]
 9 print()
10 for k in range(6):
11     print(names[k],end=" ")
12 print()
13
```

Now Try This!

Remove several names from the list. Does the program still work? Why?

```
1  # ArraySyntax05.py
2  # This program demonstrates a more flexible way to
3  # "traverse" an array by using the <len> function.
4  # NOTE: Now see what happens when you add or
5  #       remove names from the <names> array.
6
7
8  names = ["John","Greg","Maria","Heidi","Diana","David"]
9
10 print()
11 print("There are",len(names),"names in the array.")
12 print()
13
14 for k in range(len(names)):
15     print(names[k], end = " ")
16 print()
17
```

```
  ----jGRASP exec: python ArraySyntax05

There are 6 names in the array.

John Greg Maria Heidi Diana David

  ----jGRASP: operation complete.
```

```
 8 names = ["John","Greg","Maria","Heidi","Diana","David"]
 9
10 print()
11 print("There are",len(names),"names in the array.")
12 print()
13
14 for k in range(len(names)):
15     print(names[k], end = " ")
16 print()
17
```

```
  ----jGRASP exec: python ArraySyntax05

There are 6 names in the array.

John Greg Maria Heidi Diana David

  ----jGRASP: operation complete.
```

```
 8 names = ["John","Greg","Maria","Heidi","Diana","David"]
 9
10 print()
11 print("There are",len(names),"names in the array.")
12 print()
13
14 for k in range(len(names)):
15     print(names[k], end = " ")
16 print()
17
```

Now add/remove some names to/from this list and see what happens.

```
1 # ArraySyntax06.py
2 # This program demonstrates the Run-time Error
3 # that occurs if your index is too large.
4
5
6 list = [100,101,102,103,104,105,106,107,108,109]
7
8 print()
9 print(list[10])
```

Index	0	1	2	3	4	5	6	7	8	9

```
  ----jGRASP exec: python ArraySyntax06.py

Traceback (most recent call last):
  File "ArraySyntax06.py", line 9, in <module>
    print(list[10])
IndexError: list index out of range

  ----jGRASP wedge2: exit code for process is 1.
  ----jGRASP: operation complete.
```

```
1  # ArraySyntax07.py
2  # This program demonstrates a common mistake when
3  # an array is traversed with a loop.  With 10 items
4  # in the array, the indexes range from 0 to 9, but
5  # the <while> loop erroneously counts from 1 to 10.
6  # This also causes a "list index out of range" error.
7
8
9  list = [100,101,102,103,104,105,106,107,108,109]
10
11 print()
12 k = 1
13 while k <= 10:
14     print(list[k])
15     k += 1
16
```

Index	0	1	2	3	4	5	6	7	8	9

```
1  # ArraySyntax07.py
2  # This program demonstrates a common mistake when
3  # an array is traversed with a loop.  With 10 items
4  # in the array, the indexes range from 0 to 9, but
5  # the <while> loop erroneously counts from 1 to 10.
6  # This also causes a "list index out of range" error.
7
8
9  list = [100,101,102,103,104,105,106,107,108,109]
10
11 print()
12 k = 1
13 while k <= 10:
14    print(list[k])
15    k += 1
16
```

Index	0	1	2	3	4	5	6	7	8	9

```
 ----jGRASP exec: python ArraySyntax07.py


101
102
103
104
105
106
107
108
109
Traceback (most recent call last):
  File "ArraySyntax07.py", line 14, in <module>
    print(list[k])
IndexError: list index out of range

 ----jGRASP wedge2: exit code for process is 1.
 ----jGRASP: operation complete.
```

```
 1 # ArraySyntax08.py
 2 # This program demonstrates that Python will allow
 3 # you to display the entire contents of an array
 4 # with a single <print> command.  The array is
 5 # displayed in [brackets] separated by commas.
 6
 7 # NOTE: Not all languages allow you to display
 8 #       the contents of an array in this manner.
 9
10
11 names = ["John","Greg","Maria","Heidi","Diana","David"]
12
13 print()
14 print(names)
```

```
  ----jGRASP exec: python ArraySyntax08.py

['John', 'Greg', 'Maria', 'Heidi', 'Diana', 'David']

  ----jGRASP: operation complete.
```

```
 1 # ArraySyntax09.py
 2 # This program demonstrates that Python technically
 3 # does not have "Arrays".  Python has "Lists".
 4 # A list is an ordered sequence of elements.
 5 # An array is a list where all of the elements are
 6 # the same type. In spite of this, we will continue
 7 # to use Python lists as arrays.
 8
 9
10 misc = ["John","Greg","Maria","Heidi",7,3.14,True]
11
12 print()
13 print(misc)
```

```
----jGRASP exec: python ArraySyntax09.py

['John', 'Greg', 'Maria', 'Heidi', 7, 3.14, True]

----jGRASP: operation complete.
```

List Definition

A *list* is an ordered sequence of elements.

Alternate Array Definition

An *array* is a *list* where all of the elements are of the same data type.

Python Array/List Disclaimer

Python technically does not have *arrays*. Python has *lists*.

In spite of this, we will continue to use Python *lists* as *arrays* for the duration of this first year class.

```
1  # ArraySyntax10.py
2  # This program demonstrates how to create an array
3  # of a specific size filled with the same value.
4  # While most languages have an overloaded + operator,
5  # Python also has an overloaded * operator.
6
7
8  list1 = [7] * 20
9  list2 = [3.14] * 10
10 list3 = ['Q'] * 12
11 list4 = ["Hello!"] * 6
12 list5 = [True] * 10
13
14 print()
15 print(list1)
16 print()
17 print(list2)
18 print()
19 print(list3)
20 print()
21 print(list4)
22 print()
23 print(list5)
```

```
1  # ArraySyntax10.py
2  # This program demonstrates how to create an array
3  # of a specific size filled with the same value.
4  # While most languages have an overloaded + operator,
5  # Python also has an overloaded * operator.
6
7
8  list1 = [7] * 20
9  list2 = [3.14] * 10
10 list3 = ['Q'] * 12
11 list4 = ["Hello!"] * 6
12 list5 = [True] * 10
13
14 print()
15 print(list1)
16 print()
17 print(list2)
18 print()
19 print(list3)
20 print()
21 print(list4)
22 print()
23 print(list5)
```

```
----jGRASP exec: python ArraySyntax10.py

[7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7]

[3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14, 3.14]

['Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q', 'Q']

['Hello!', 'Hello!', 'Hello!', 'Hello!', 'Hello!', 'Hello!']

[True, True, True, True, True, True, True, True, True, True]

----jGRASP: operation complete.
```

```
 1 # ArraySyntax11.py
 2 # This program demonstrates that 2 arrays
 3 # can be combined to create a third, larger
 4 # array using the overloaded + operator.
 5 # This is very similar to "string concatenation".
 6
 7
 8 listA = [11,22,33,44,55]
 9 listB = [66,77,88,99]
10
11 listC = listA + listB
12
13 print()
14 print(listA)
15 print(listB)
16 print(listC)
17
```

```
 1 # ArraySyntax11.py
 2 # This program demonstrates that 2 arrays
 3 # can be combined to create a third, larger
 4 # array using the overloaded + operator.
 5 # This is very similar to "string concatenation".
 6
 7
 8 listA = [11,22,33,44,55]
 9 listB = [66,77,88,99]
10
11 listC = listA + listB
12
13 print()
14 print(listA)
15 print(listB)
16 print(listC)
17
```

```
----jGRASP exec: python ArraySyntax11


[11, 22, 33, 44, 55]
[66, 77, 88, 99]
[11, 22, 33, 44, 55, 66, 77, 88, 99]


----jGRASP: operation complete.
```

```
1 # ArraySyntax12.py
2 # This program demonstrates that the overloaded
3 # += operator behaves the same way with arrays
4 # as it does with strings.
5
6
7 list = [11,22,33,44,55]
8
9 print()
10 print(list)
11
12 list += [66,77,88,99]
13
14 print()
15 print(list)
```

```
1  # Ar
2  # Th
3  # +=
4  # as
5
6
7  list = [11,22,33,44,55]
8
9  print()
10 print(list)
11
12 list += [66,77,88,99]
13
14 print()
15 print(list)
```

```
----jGRASP exec: python ArraySyntax12.py

[11, 22, 33, 44, 55]

[11, 22, 33, 44, 55, 66, 77, 88, 99]

----jGRASP: operation complete.
```

Slicing

When working with arrays in Python, a *slice* refers to a piece of an array.

list Array

Index	0	1	2	3	4	5	6	7	8	9
Value	100	101	102	103	104	105	106	107	108	109

↑ `list[3:8]` *slice* ↑

```
1 # ArraySyntax13.py
2 # This program demonstrates how to display a
3 # "slice" of an array using the [start:stop]
4 # syntax.  The array is displayed starting on
5 # first index and stopping just BEFORE the second.
6
7
8 list = [100,101,102,103,104,105,106,107,108,109]
9
10 print()
11 print(list)
12
13 print()
14 print(list[3:8])
15
```

```
----jGRASP exec: python ArraySyntax13.py

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

[103, 104, 105, 106, 107]

----jGRASP: operation complete.
```

```
 7
 8 list = [100,101,102,103,104,105,106,107,108,109]
 9
10 print()
11 print(list)
12
13 print()
14 print(list[3:8])
15
```

```
1  # ArraySyntax14.py
2  # This program demonstrates that if you leave
3  # out the first number in a slice, it will
4  # start at index 0 by default.
5
6
7  list = [100,101,102,103,104,105,106,107,108,109]
8
9  print()
10 print(list)
11
12 print()
13 print(list[0:8])
14
15 print()
16 print(list[:8])
```

```
----jGRASP exec: python ArraySyntax14.py

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

[100, 101, 102, 103, 104, 105, 106, 107]

[100, 101, 102, 103, 104, 105, 106, 107]
```

```
 7 list = [100,101,102,103,104,105,106,107,108,109]
 8
 9 print()
10 print(list)
11
12 print()
13 print(list[0:8])
14
15 print()
16 print(list[:8])
```

```
1  # ArraySyntax15.py
2  # This program demonstrates that if you leave
3  # out the second number in a slice, it will
4  # go all of the way to the end of the array.
5  # It also shows that it if the second number
6  # is way too large, you do not get an error.
7  # Instead, it also goes to the end of the array.
8
9
10 list = [100,101,102,103,104,105,106,107,108,109]
11
12 print()
13 print(list)
14
15 print()
16 print(list[3:])
17
18 print()
19 print(list[3:100])
```

```
----jGRASP exec: python ArraySyntax15.py

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

[103, 104, 105, 106, 107, 108, 109]

[103, 104, 105, 106, 107, 108, 109]
```

```
 9
10 list = [100,101,102,103,104,105,106,107,108,109]
11
12 print()
13 print(list)
14
15 print()
16 print(list[3:])
17
18 print()
19 print(list[3:100])
```

```
# ArraySyntax16.py
# This program was originally program GraphicsLibrary20.py
# from Chapter 6.  Look closely at the <fillPolygon> commands.
# The reason the [brackets] are required is that we are actually
# passing an "array of integers" to the <fillPolygon> procedure.
# This means you have been using arrays for a while now.


from Graphics import *

beginGrfx(1000,650)

drawCircle(500,100,50)
drawLine(500,150,500,400)
drawLine(500,400,400,600)
drawLine(500,400,600,600)
drawLine(300,225,700,225)
setColor("blue")
fillPolygon([425,375,425,425,350,550,450,600,
500,450,550,600,650,550,575,425,575,375])
setColor("red")
fillPolygon([350,200,650,200,650,250,575,250,
575,350,425,350,425,250,350,250])

endGrfx()
```