MCIS 5313 - Data Structure and Algorithms

Week 1

Dr. Ahmad Al Shami
aalshami@saumag.edu

Aug 10, 2020

Overview

- Syllabus
- Books
- 1. Data Structure and algorithms in Python, Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser- *2013 Wiley.*
- 2. Data Structure and algorithms Using Python, Rance D. Necaise – Wiley 2011.
- 3. The Design & Analysis of Algorithms, 3rd Edition, Anany Levitin

- Compilers
  - Online Repl.it – Choose python3
  - Installing Python and PyCharm Plugin (instruction available)

- Pythyon - Fundamentals

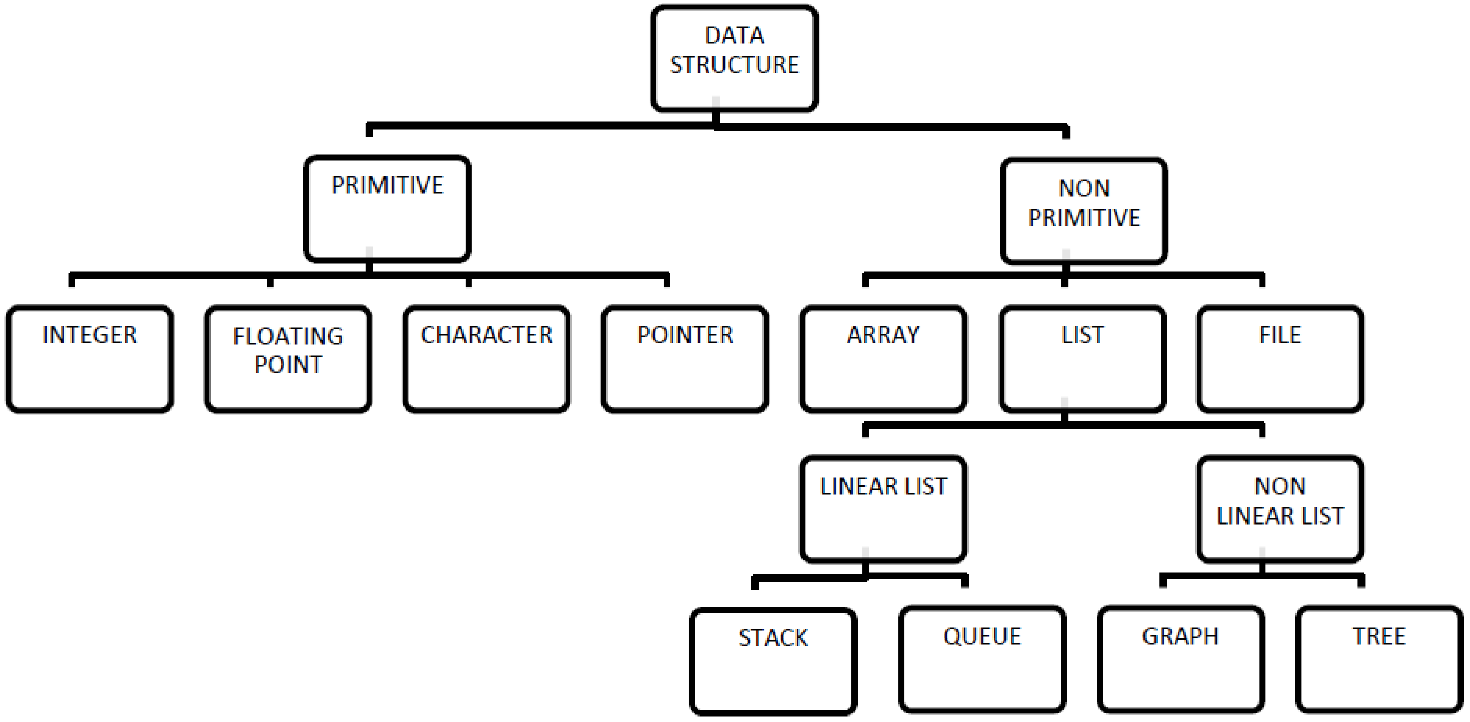
Data Structure

In computer science, a **data structure** is a data organization, management and storage format that enables efficient access and modification.[1][2][3] More precisely, a **data structure is a collection of data values**, the **relationships among them**, and the functions or operations that can be applied to the data.[4]

Algorithms

In mathematics and computer science, an algorithm (/ˈælɡərɪðəm/ (listen)) is an unambiguous specification of how to solve a class of problems. Algorithms can perform calculation, data processing and automated reasoning tasks.

Classification of Data Structure



Install Python your development environment

Installing Python and PyCharm on Windows

Please find accompanied .pdf document

Advantages of Using Python

- **Readability**
- Python programs use clear, simple, and concise instructions that are easy to read even by those who have no substantial programming background. Programs written in Python are, therefore, easier to maintain, debug, or enhance.

- **Higher productivity**
- considerably shorter, simpler, and less verbose than other high level languages
- it has well-designed built-in features and standard library

Advantages of Using Python

- **Less learning time**
- Python is relatively easy to learn. Many find Python a good first language for learning programming because it uses simple syntax and shorter codes

- **Runs across different platforms**
- Python works on Windows, Linux/UNIX, Mac OS X, other operating systems and small-form devices.

Installing Python in windows

- Installing Python in Windows
  - https://www.python.org/downloads/

- Installing Python in Mac
  - https://www.python.org/downloads/mac-osx/

Keywords *Python Syntax*

- Keywords

and	assert	break	class
continue	def	del	elif
else	except	exec	finally
for	from	global	if
import	in	is	lambda
not	or	pass	print
raise	return	try	while
with	yield		

Identifiers

- Identifiers is a given to a

  - function, class, variable, module, or other objects
  - Identifier can be combination of letter, lowercase, underscores, and digits(0-9).
  - Special characters such as %, @, and $ not allowed
  - A identifier should not begin with a number
  - You cannot not use Python keywords as identifiers

Variables

- Variables in other languages
  - int x, y; double x, y;   type should be declared

- In python, handling variables are more flexible
  - You can declare a variable by giving its value

```
>>> my_variable = 10
>>> my_variable = my_variable +3
>>> print(my_variable)

>>> my_varialble="yellow"
>>> print(my_variable)
```

String

- Sequence of Unicode characters

```
>>> stringA = “I am enclosed in single quotes”
>>> stringB=“I am enclosed in double quotes”.
>>> s=“Hello Python”
```

-12	-11	-10	-9	-8	-6	-6	-5	-4	-3	-2	-1
H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s[0]
>>>s[len(s)-1]
>>>s[-1]
```

Concatenating and Repeating Strings

```
>>> “Hello” + “Python”
‘HelloPython’
```

Entering “**^**” * 5 will yield:

```
‘**^****^****^****^****^**’
```

```
>>>s = “**^**”
>>>s * 5
‘**^****^****^****^****^**’
```

Concatenating and Repeating Strings

```
>>> “Hello” + “Python”
‘HelloPython’
```

Entering “**^**” * 5 will yield:

```
‘**^****^****^****^****^**’
```

```
>>>s = “**^**”
>>>s * 5
‘**^****^****^****^****^**’
```

Lists

- **Define lists**

```
>>> my_list = [item_1, item_2, item_3]
>>> colors = [“red”, “orange”, “yellow”, “green”, “indigo”, “white”]
>>> print(colors[0])
Red
>>> print(colors[4])
indigo
>>> print(colors)
[‘red’, ‘orange’, ‘yellow’, ‘green’, ‘indigo’, ‘white’]
>>> colors.append(“violet”)
>>> print(colors)
[‘red’, ‘orange’, ‘yellow’, ‘green’, ‘indigo’, ‘violet’]
```

Basic operators

+	Addition	adds the value of the left and right operands
-	Subtraction	subtracts the value of the right operand from the value of the left operand
*	Multiplication	multiplies the value of the left and right operand
/	Division	divides the value of the left operand by the right operand
**	Exponent	performs exponential calculation
%	Modulus	returns the remainder after dividing the left operand with the right operand
//	Floor Division	division of operands where the solution is a quotient left after removing decimal numbers

Examples

Addition: >>>1 + 3 4 Subtraction: >>>10 – 4 6 Multiplication: >>>4 * 2 Division: >>>10 / 2 5.0	Exponent >>>2**3 2 raised to the power of 3 8 Modulus >>>17 % 5 2 Floor Division *Floor division, on the other hand, returns the quotient after removing fractional numbers:* >>>17 // 5 3

Code examples

```
meal = 65.50
tax = 6.6 / 100
tip = 20 / 100
meal = meal + meal * tax
total = meal + meal * tip

>>>total
83.78760000000001
```

Assignment Operators

Operators	Function
=	assigns the value of the right operand to the left operand
+= add and	adds the value of the right and left operand and assigns the total to the left operand
-= subtract and	deducts the value of the right operand from the value of the left operand and assigns the new value to the left operand
*= multiply and	multiplies the left and right operand and assigns the product to the left operand
/= divide and	divides the left operand with the value of the right operand and assigns the quotient to the left operand
**= exponent	performs exponential operation on the left operand and assigns the result to the left operand
//= floor division and	performs floor division on the left operand and assigns the result to the left operand

Relational Operator

Operator	Meaning
==	is equal to
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
!=	is not equal to

Relational operator Examples

Examples:

```
>>> 8 == 6+2
True
>>> 6 != 6
False
>>> -1 > 0
False
>>> 7 >= 5
True
```

Logical Operator

- and, or , not

```
>>> (8>9) and (2<9)
False

>>> (2>1) and (2>9)
False

>>> (2==2) or (9<20)
True

>>> (3!=3) or (9>20)
False

>>> not (8 > 2)
False
```

Conditional Statements

- if-then-else statement

```
if condition1:
block1_statement
elif condition2:
block2_statament
else:
block3_statement
```

Conditional Statements examples-1

```
def your_choice(answer):
    if answer > 5:
    print(“You are overaged.”)
    elif answer <= 5 and answer >1:
    print(“Welcome to the Toddler’s Club!”)
    else:
    print(“You are too young for Toddler’s Club.”)
    print(your_choice(6))
    print(your_choice(3))
    print(your_choice(1))
    print(your_choice(0))
```

Conditional Statements examples-2

```
def your_choice(answer):
    if answer > 5:
    print(“You are overaged.”)
    elif answer <= 5 and answer >2:
    print(“Welcome to the Toddler’s Club!”)
    elif answer == 2:
    print(“Welcome! You are a star member of the Toddler’s Club!”)
    else:
    print(“You are too young for Toddler’s Club.”)
    print(your_choice(6))
    print(your_choice(3))
```

For Loop in Python

```
pizza = [“New York Style Pizza”, “Pan
Pizza”, “Thin n Crispy Pizza”, “Stuffed
Crust Pizza”]

for choice in pizza:

if choice == “Pan Pizza”:
print(“Please pay $16. Thank you!”)
print(“Delicious, cheesy ” + choice)
else:
print(“Cheesy pan pizza is my all-time
favorite!”)

print(“Finally, I’m full!”)
```

```
Delicious, cheesy New York Style Pizza
Please pay $16. Thank you!
Delicious, cheesy Pan Pizza
Delicious, cheesy Thin n Crispy Pizza
Delicious, cheesy Stuffed Crust Pizza
Cheesy pan pizza is my all-time favorite!
Finally, I’m full!
```

While loop

while condition
Statement
 	
counter = 0
while (counter < 10):
print(‘The count is:’ , counter)
counter = counter + 1
	The count is: 0
	The count is: 1
	The count is: 2
	The count is: 3
	The count is: 4
	The count is: 5
	The count is: 6
	The count is: 7
	The count is: 8
	The count is: 9
	Done!

While loop

```
while condition
Statement
```

```
counter = 0
while (counter < 10):
print(‘The count is:’ , counter)
counter = counter + 1
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
The count is: 9
Done!
```

User Defined function-1

- A function is a set of statements that perform a specific task, a common structuring element that allows you to use a piece of code repeatedly in different parts of a program.

- A user-defined Python function is created or defined by the def statement and follows the syntax:

- def function_name (parameter list);

  function body/statemenst

User Defined function-2

```
def love_pizza():
print “I love Pizza!”
```

- Function with a parameter and return keyword:

```
def absolute_value(number):
if number >= 0:
return number
else:
return -number

print(absolute_value(3))
print(absolute_value(-5))
```

```
def shutdown(yn):
if yn.lower() == “y”:
return(“Closing files and shutting down”)
elif yn.lower() == (“n”):
return(“Shutdown cancelled”)
else:
return(“Please check your response.”)

print(shutdown(“y”))
print(shutdown(“n”))
print(shutdown(“x”))
```

User Defined function-3

- More than two parameters

```
def calculator(x, y):
return x * y + 2

print(calculator(2,6))
print(calculator(3,7))
```

Run the code and you'll get the output:

```
14
23
```

* Function can call other functions

```
def members_total(n):
return n * 3

def org_total(m):
return members_total(m) + 5


print(org_total(2))
print(org_total(5))
print(org_total(10))
```

Further Reading

The Ultimate Beginner's Guide –Andrew Johansen