Humna Sultan

Essentials of CS115 (Fall 2022, Prof. Akcam)

Intro to CS in Python with a (very!) big emphasis on recursion

9/7/22

Basics

- There are many programming languages - not all are very useful / effective for use
- Programming languages evolve and change - different ones are popular / practical at different times
- Huffman Data Compression
    - Compresses commonly referenced items with short codes
    - Compresses less referred items with longer codes
- In code
    - 5/2 = 2.5
    - 5//2 = 2
        - // is integer division
    - 5**2 = 25
        - ** is exponent
- Precedence order - like PEMDAS in coding

1. ()
2. **
3. * / //
4. + -

EXERCISE 1

- Average of 32, 12, and 46 in code
    - ```
      >>> print((32+12+46)/3)
      30.0
      ```
    - print(32+12+46/3) is wrong because it will do 46/3 and then add

EXERCISE 2

- Write the math problem in IDLE
    - ```
      >>> print((4*(8-5))/((2**3)-7))
      12.0
      ```

Variables

- Variables hold values - you can use variables to hold numbers instead of using the actual numbers themselves
- You cannot print or use a variable that is not defined

```
>>> number_rooms=3
>>> number_rooms
3
>>> print(number_rooms*3)
9
>>> number_rooms = 5
>>> print(number_rooms*3)
15
>>> print(nr_bathrooms)
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    print(nr_bathrooms)
NameError: name 'nr_bathrooms' is not defined
>>>
```

```
>>> x = 2
>>> y = 3
>>> z = 5
>>> x, y, z = 2, 3, 5
>>> number, number2, number3m 2, 3, 5
    SyntaxError: invalid syntax
>>> number, number2, number3 = 2, 3, 5
```

- += adds to the variable on the left (shortcut)
  - EX (x is originally 2):

```
>>> x = x + 2
>>> print(x)
4
>>> x+=2
>>> print(x)
6
```

String Variables

- In python, it is okay to use EITHER single or double quotes when defining a string
- Each character in a string is an index - starts at 0
  - EX: in the String “Test”, T is index 0, e is index 1, s is index 2, t is index 3
    - Length of string is 4, but length-1 gives last index
- Function len() gives you the length of a string
- Finding last letter in a string
  - print(word[len(word)-1]) gives index of last letter
  - print(word[-1]) also gives index of last letter

```
>>> word = "test"
>>> word = 'test'
>>> print(word[0])
t
>>> print(word[len(word)-1])
t
>>> print(word[-1])
t
>>> len(word)
4
>>> word = 'Computer'
>>> len(word)
8
>>>
```

9/9/22

- Substring Example
    - String “Computer”
        - [0] = “C” ; [1] = “o” ; [2] = “m” ; [3] = “p”
        - 8 characters ; max index is 8-1 = 7
- Last index of a string is len(string)-1
- String Indexes Example
    - course = “Computer
    - uni = “Stevens”
        - uni[3] = “v”
        - uni[5] = “n”
        - uni[len(uni)-2] = “n”
        - course[0] = “C”
        - course[len(course)] = undefined (course[8] does not exist)
        - uni[-3] = “e”
        - uni[-len(uni)] = “S”
- Making functions
    - “def” is a keyword indicating a function
        - After “def” is the function name
    - Inside the parentheses after the function name, there are inputs for use
    - There is a colon placed after the parentheses (input declaration)
    - *** The line declaring the function is referred to as the “function header”
    - Code placed inside functions is tabbed forward (known as “function body”)
        - Indentation is necessary for function body content
    - Call a function using a function’s name and parameters inside (known as “function call”)
- Comments
    - #this is a comment
    - “”” this is also a comment “”””
- Composition of functions
    - Placing a function within the parameters of another so that the value of one becomes the parameter of the other
    - EX: return dbl(dbl(x))
        - The one inside will happen first and the value will become the parameter of the outside one
- Lists - can hold multiple items inside of them
    - listname = [xx, yy, zz]
        - xx is index 0, yy is index 1, zz is index 2
    - listname = [] is empty list
    - If you print a list, there will be square brackets printed to show you that it is a list
    - You can reference specific items in a list using index numbers
    - fruits[len(fruits)] is undefined because the max index is length - 1
    - You can also use negative indexing in lists!!

        - Negative indexing starts at the last character (index -1 is the last character)
    - You can reference a portion of a list
        - Syntax: listname[starting index: ending index]
        - EX: elements[1:3] will only
            - Includes index 1, does not include index 3
        - You can also do listname[:2]
            - Would start at 0 and go until 2 (not including 2)
        - You can also do listname[2:]
            - Would start at 2 and go until the end of the list
        - listname[:-2] would start at 0 and go until index -2 (not including index -2)

9/12/22

- If you want to concatenate to a list, you have to reassign it + that value
    - EX:
        - L + [50] does not keep the 50 in L
        - L = L + [50] does change the actual variable
            - You can also do L += 50
    -

```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L+10
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    L+10
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L = L + [50]
>>> L
[1, 42, 3, 4, 50]
>>> x = 4
>>> x+2
6
>>> x = x + 2
>>> x
6
>>> L * 2
[1, 42, 3, 4, 50, 1, 42, 3, 4, 50]
>>> L += 50
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    L += 50
TypeError: 'int' object is not iterable
>>> L += [50]
>>> L
[1, 42, 3, 4, 50, 50]
```

- You can imbed a list within a list
  - ```
    >>> M = [42, "Hello", 3+2j, 3.141, [1,2,3,4,5,6]]
    >>> M[4]
    [1, 2, 3, 4, 5, 6]
    ```
- Indexing and slicing in lists (L = [1, 42, 3, 4, 50, 50])
  - L[0:3] - 0 is start index and 3 is end index (0 is included, 3 is not)
  - L[0:3:2]
    - 0 is start index, 3 is end index (not included), 2 is step/counter
      - Counts with 0+2, 0+4, etc.
    - Will only print 0 and 2 places
  - L[1:]
    - Will print index 1 until the end of the list (including the end)
  - L[:-1]
    - Cuts off the last item in the list
  - L[1:-2]
    - [42, 3, 4]
- You can also index and slice lists
  - EX:
    - ```
      >>> S = "I love Spam!"
      >>> S[12:6:-1]
      '!mapS'
      ```
    - ```
      >>> S[12:6:-3]
      '!p'
      ```
- Value returning function: returns a value back
- Void function: does not return a value

- Some built-in functions!!
  - list(map(function to apply to all indexes, list of elements))
    - print(list(map(function to apply to all indexes, list of elements)))
  - range()
    - range(10) - 10 is the end index (excluded)
      - Any value from 0 to 10 will be included (10 is excluded because it is the end index)
    - range(2, 10) - 2 is start value (included) and 10 is the end index (excluded)
    - range(2, 10, 2) - 2 is the start value, 10 is end index, and it counts by 2 each time
      - For numbers 1-10 - 2, 4, 6, 8 are returned
      ```

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2,10))
[2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2,10,2))
[2, 4, 6, 8]
>>> list(range(7,2,2))
[]
>>>
```

- If start index is larger than end index, it is empty unless counting backwards

```
>>> list(range(7,2,-2))
[7, 5, 3]
>>>
```

9/14/22

<u>Some more built-in manipulative functions 🙂</u>

- map() function
  - Does not actually update the list variable unless you re-assign it
    - DOES NOT update

```
L = [1,2,3,4]
print(list(map(dbl, L)))
print(L)
```

    - DOES update

```
L = [1,2,3,4]
L = list(map(dbl, L))
print(L)
```

- reduce() function
  - Must be imported to use - “from functools import reduce” at top

```
def add(x,y):
    return x+y

print(reduce(add, [1,2,3,4]))
```

    - Adds 1+2, then that sum +3, and then that sum+4
      - Returns 10
    - Recurses through each element
- min(x,y) and max(x,y)
  - Return the minimum and maximum values
- range() function
  - range(5+1) gives 1, 2, 3, 4, 5 (including 5)
  - range(5) gives 0, 1, 2, 3, 4 (excluding 5)

map()	reduce()

- No need for package import
- Takes two arguments
- Takes 1 argument takes receives 1 argument and returns 1 result
  - Function for application has 1 argument
- Map returns a list (use list())
	- Must import functools
	- Takes two arguments
	- Takes receives 2 arguments return 1 result
	  - Function for application has 2 arguments
	- Returns integer value

Intro to Booleans!

- == checks whether something is true or false
- “spam” > 42 returns true in Python 2.x

```
>>> 3 == 1+2
True
>>> 42 == 'spam'
False
>>> 'spam' > 42
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    'spam' > 42
TypeError: '>' not supported between instances of 'str' and 'int'
>>>
```

- False represents value 0; True represents value 1
  - 2**False == True is True
  - 2*False == True is False

9/16/22

- If-else statements
  - Serve to check conditions
  - Elif is “else if”
    - You can use it multiple times within one statement
  - Indented parts of code go under the line above
    - You have indented lines under each part of the loop
  - Depending on the conditions, different indented parts will run
    - elif(s) is/are only checked if “if” is false
      - If all conditions are false, then else indented code runs

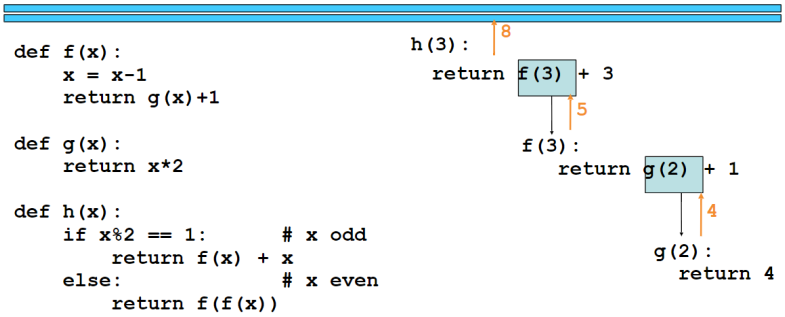
- filter() function
  - Checks a condition for all values in a list and returns values that made the condition true
- modulus (mod) operation
  - Finds the remainder after a division operation
  - EX: 10 % 3 = 1

Humna Sultan

9/19/22

Recursion <3

Examples:



h(3) → f(3) + 3 → g(2) + 1 → g(2) = 4 → g(2) + 1 = 5 → 5 + 3 = 8 → h(3) = 8

- EX: factorial function : n! = n*(n-1)*(n-2)...*1
  - 0! = 1
  - Base case:
    - if n == 0: return 1
      - 0! = 1 because any other value will change the answer
  - Recursive case:
    - else: n*factorial(n-1)
  - EX: factorial(5)
    - 5*factorial(4) → 4*factorial(3) → 3*factorial(2) → 2*factorial(1) → 1*factorial(0) = 1*1 = 1
      - 1*1 = 1 → 2*1 = 2 → 3*2 = 6 → 4*6 = 24 → 5*24 = 120
- “A Tower of Fun” slide
  - Base case:
    - if n == 1: return 2
  - Recursive case:
    - tower(n-1)**2
  - Explanation: tower(4) → tower(3)*2 → tower(2)**2 → tower(1)**2 → tower(1) = 2

Humna Sultan

9/21/22

- Create a len function that recursively finds the length of an list

```
def len(lst):
    if list == [ ]:
        return 0
    return 1 + len(lst[1:])
```

- Examples with function
  - lst [ ] → 0
  - lst [1, 2] → 1 + len([2]) → 1 + len(lst[ ]) → 1 + 1 + 0 = 2
  - lst[1,2,[3,4]] → 1 + len([2,[3,4]]) → 1 + len([[3,4]]) → 1 + len([ ]) → 1+1+1+0 = 3
    - Need len(lst[0]) to take the list [3,4] and count those too

★ filter() filters through a list with a condition

Example of Embedded Function

```
def divides(n):
    def div(k):
        return n%k == 0
    return div
```

- Assigns n and then k and the result
- EX:

```
f = divides(15)
print(f(3)) → prints True
```

Instead of…

```
def dbl(x):
    return 2*x
```

You can do…

```
dbl = lambda x : 2*x
```

EX with lambda and filter:

```
filter(lambda x : x % 2 == 0, range(100))
```

Humna Sultan

9/26/22 + 9/28/22

Use-It, Lose-It Problem:

Problem

Given a capacity and a list of positive-number items, write a function named subset that returns the largest sum that can be made from the items without exceeding the capacity.

Capacity 42 lbs

- Base case: capacity <= 0 or items == [ ]
    - Both variables must be checked to see if they are empty
- if item[0] > capacity
    - return subset(capacity, items[1:])
- else:
    - useIt = items[0] + subset(capacity-item[0], items[1:])
        - Took item with you and took it out of capacity
    - loseIt = subset(capacity, items[1:])
        - Did not take item with you and capacity stays the same
    - return max(useIt, loseIt)

<u>Example Using Subsets</u>

- subset(7, [2, 3, 4])
    - → 2 + subset(5, [3, 4]) = 6
        - → 3 + subset(2, [4])
            - → subset(2, [0]) = 0
        - → subset(5, [4]) = 4

- 6 is greater than 4

- subset(7, [3,4])
    - 3 + subset(4, [4])
        - 4 + subset(0, [ ])
        - subset(4, [ ]) = 0
    - subset(7, [4])

- Both subsets are the same

★ Each time you recurse, you change/check useIt and loseIt both
★ Reviewed Knapsack Lab and EditDistance Lab in this class 🙂

9/30/22

```
def addTwoDigits(n):
    return n // 10 + n % 10

def largestNumber(n):
    return n**10-1

def reverse(L):
    return L[::-1]
```

:: accepts the default for the start and end of a list

- Default start is 0
- Default end is len(list) - 1

Higher Order Functions

- map(function, x) → x = list(iterator), you need to put "list()" around the whole map function call because otherwise it returns the object (not a list)
    - Takes one parameter, returns 1 value
- reduce(function, list)
    - Takes two parameters, returns 1 value
- filter(boolean function, list) → list(iterator), you need to put "list()" around the whole map function call because otherwise it returns the object (not a list)
    - Has boolean function

<u>Examples of using functions</u> <3

list = [1, 3, 6]
Calculate : (1^2 + 3^2 ^ 6^2) % 7

```
newL = list(map(lambda x : x ** 2, list))
result = reduce(lambda i, n : i * n, newL)
print(list(filter(lambda x : x % 7 == 0, [result] )))
```

<u>Recursive Sum Function</u>

```
def mySum(L):
    if L == [ ]:
        return 0
    else:
        return L[0] + mySum(L[1:])
```

Reverse a String Functions

```
def reverseString(string):
        if string == “”:
                return “”
        else:
                return string[-1] + reverseString(string[:-1])
```

OR

```
def reverseString(string):
        if string == “”:
                return “”
        else:
                return reverseString(string[1:]) + string[0]
```

10/5/22

★ Reviewed Longest Common Subsequence Lab

Recursive Factorial

```
def factorial(n):
        if n == 0:
                return 1
        else:
                 return n * factorial(n-1)
```

factorial(3)
→ 3 * factorial(2)
→ → 2 * factorial(1)
→ → → 1 * factorial(0)
→ → → → return 1

- Stored in program as a stack

10/7/22

```
def factorial (n, a = 1):
#default a value is 1
```

★ Reviewed reversing a String recursively
★ In a tail recursive, there is only the function call and no operations before or after

Tuples, Lists, Dictionaries

() - used for tuples
- Immutable (cannot be changed)

[ ] - used for lists
- Mutable (can be changed)

{ } - used for dictionary

Examples of Using Tuples, Lists, Dictionaries

```
foo = (42, “hello”)
foo[0] = 55  #ERROR (you cannot change the value of a tuple!)

foo = [42, “hello”]
foo[0] = 55 #does work!

D = { }
D =[‘ran’] = ‘spam’
>>> D
[‘ran’ : ‘spam’]
```

‘ran’ is the key, ‘spam’ is the value

10/11/22 + 10/12/22

- Tuple
  - Immutable structures
  - Set of values in parentheses separated by commas
  - a = (42, ‘3’, ‘hello’)
    - a[0] IS 42
    - a[1] = 25 IS AN ERROR (immutable)
- List
  - Mutable structures
  - Set of values in square brackets separated by commas
  -
- Dictionary
  - Mutable structures
  - Set of values in curly braces with key-value pairs
  - b = {‘a’ : b, ‘b’ :10}
    - print(b[‘b’]) IS 10

Humna Sultan

Examples of Using the 3 <3

```
>>> t = 1234, 4567, 'hello'
>>> #tuples do not need paranthesis with initialization
>>> print(t)
(1234, 4567, 'hello')
>>> x, y, z = 1234, 5678, 'hello'
>>> #declares 3 different variables with respective values in order
>>> t[0] = 12345
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    t[0] = 12345
TypeError: 'tuple' object does not support item assignment
>>> v = ([1,2,3], [3,4,5])
>>> print(v)
([1, 2, 3], [3, 4, 5])
>>> #makes mutable lists inside of a tuple
>>> v[0] = [6,7]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    v[0] = [6,7]
TypeError: 'tuple' object does not support item assignment
>>> #lists inside of a tuple are NOT mutable
>>> #tuple overrides the mutable ability of the lists
>>> t
(1234, 4567, 'hello')
>>> u = t, (1, 2, 3, 4, 5)
>>> u
((1234, 4567, 'hello'), (1, 2, 3, 4, 5))
>>> #embedded tuples
>>> t
(1234, 4567, 'hello')
>>> empty = ()
>>> empty
()
>>> #empty tuple
>>> singleton = (1234)
>>> singleton
1234
>>> #not a tuple
```

```
>>> singleton = 1234,
>>> singleton
(1234,)
>>> #a tuple! - use a comma to indicate a tuple with a single item
>>> x, y, z = t
>>> x
1234
>>> y
4567
>>> z
'hello'
>>> #if you know the length of the tuple, you can set variables for each value in the tuple
>>> print(f'x = {x}, y = {y}, z = {z}')
x = 1234, y = 4567, z = hello
>>> #printing format!!
```

```
>>> d = {}
>>> d
{}
>>> #empty dictionary
>>> d['jack'] = 4098
>>> d
{'jack': 4098}
>>> # ' jack' is key for value 4098
>>> d = {'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> d
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> d = dict([('jack', 4098), ('sape', 4139), ('guido', 4127)])
>>> d
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> #multiple different ways to create a dictionary
>>> d['jack'] = 6567
>>> d
{'jack': 6567, 'sape': 4139, 'guido': 4127}
>>> #mutable!
>>> #if the key is in the dictionary, the value will be updated; if not in the dictionary, a new key-value pair will be created
>>> 'jack' in d
True
>>> #tells us if a key is present in the dictionary
>>> 'zumrut' in d
False
>>> 'zumrut' not in d
True
>>> del d['jack']
>>> d
{'sape': 4139, 'guido': 4127}
>>> #you can use "del" keyword to delete a key-value pair
>>> d.items()
dict_items([('sape', 4139), ('guido', 4127)])
>>> # .items() gives all key-value pairs
>>> d.keys()
dict_keys(['sape', 'guido'])
>>> d.values()
dict_values([4139, 4127])
>>> # .keys() prints all keys ; .values() prints all values
```

```
>>> values = list(d.values())
>>> values
[4139, 4127]
>>> #makes a list with all of the values in the dictionary
```

Memoization

- A technique of the recording of the intermediate results so that they can be used to avoid repeated calculation and speed up programs
    - Avoids the repetition in recursion
        - Optimizes recursion!!
- FIRST STEP IN EVERY MEMOIZATION PROBLEM: check if the value has been calculated already

Kung Fu Panda Problem

- 1 step at a time
    - 1 way to do this
        - (1, 1, 1)
- 2 steps at a time
    - 2 different ways to do this
        - (1, 2) (2, 1)
- 3 steps at a time
    - 4 different ways to do this
        - (1, 1, 1) (1, 2) (2, 1) (3)
- 4 steps at a time

- 7 different ways to do this
    - (1, 1, 1, 1) (1, 1, 2) (1, 2, 1) (2, 1, 1) (2, 2) (3, 1) (1, 3) (4)

Code for Panda Problem

```
def f(n):
    if n == 1:
        return 1
    # 1 + 1 + 1 is only way - return 1
    elif n == 2:
        return 2
    # you can do either 1 + 2 or 2  + 1 - return 2
    elif n == 3:
        return 4
    else:
        return f(n-3)+f(n-2+f(n-1)
```

10/17/22 + 10/19/22

Number Bases

DECIMAL EX: 4312 = 4 * 10^3 + 3 * 10^2 + 1 * 10^1 + 2 * 10^0

BASE 20 EX: 132 in base 20 = 1 * 20^2 + 3 * 20^1 + 2 * 20^0
= 400 + 60 + 2 = 462

★ The rightmost digit in every base represents the 1s place

5 in base 10 in different bases:
BASE 2: 101
BASE 3: 12
BASE 4: 11
BASE 5: 10
BASE 6: 5
- 5 of 1s place

BASE 42: 5
- Anything above base 5 representing 5 in base 10 is just 5

- Increasing the base makes the result shorter
    - EX: 10^9 in base 20 is shorter than that value in base 1 or base 2
- What's the ratio between the lengths of a number in bases x and y?

Humna Sultan

Converting between Bases Examples

36 base 10 to base 2:
32 is the highest power in 2
4 is remaining → 2^2
100100

1001 in base 2 to base 10:
First 1 = 1
2^3 = 8
8 + 1 = 9

1101 in base 2 to base 10:

25 in base 10 to base 2:
11001
2^4 = 16
25 - 16 = 9
9 = 2^0 + 2 ^3

110011 in base 2 to base 10:
32 + 16 + 2 + 1 = 51

- "Left shifting" = multiply by 10
- "Right shifting" = divide by 10

★ Calculate rightmost using 25 % 2
  - Divide 25 // 2 (integer division) to keep moving

EX: 25 in decimal to base 2 → 25 %2 = 1 → 25 // 2 = 12 → 12 % 2 = 0 → 12 // 2 = 6 → 6 % 2 == 0 → 6 // 2 = 3 → 3 % 2 = 1 → 3 // 2 = 1 → 1 % 2 = 1 → 1 // 2 = 0

Humna Sultan

10/21/22

Russian Peasants

21 and 6
- Divide 21 by 2, multiply 6 by 2

21 // 2 = 10
6*2 = 12
10 // 2 = 5
12 * 2 = 24
5 // 2 = 2
24 * 2 = 48
2 // 2 = 1
48 * 2 = 96
- Add all the pairs of the odd values of 21 // 2

96 + 24 + 6 = 126!
21 * 6 = 126!

27 and 13
27 // 2 = 13
13 * 2 = 26
13 // 2 = 6
26 * 2 = 52
6 // 2 = 3
52 * 2 = 104
3 // 2 = 1
104 * 2 = 208

13 + 26 + 104 + 208 = 351
27 * 13 = 351

★ Right digit is always 1 in a binary odd number
★ Choose the smaller number when dividing (green numbers)

Two's Complement
- For getting negative binary numbers

1. Find the binary representation of the positive number
2. flip/reverse/switch every digit of that number (0 becomes 1, 1 becomes 0)
3. Add 1 to the number you found at step 2

Humna Sultan

Two's Complement Example

Representation of -3 base 10

1. 0011 is +3 in decimal
2. 0011 → 1100
3. 1100 + 1 = 1101 base 2 = -3 base 10

1 | 101
-8 | +4 + 1 = -3
Leftmost digit of 1 represents sign (value of -8)

- ★ The leftmost digit that we have in binary represents the sign of the binary number
  - Leftmost digit of 1 = negative
  - Leftmost digit of 0 = positive
- ★ Leftmost digit is known as "sign bit" - represents the sign

- ★ One's complement is the first 2 steps of two's complement

Thinking in 3 Bits!

- You cannot have a negative of 0
- Solve for -1: +1 in base 10 is 001 in base 2 → 110 → 110 + 1 = 111 in base 2
- Largest positive number: 011
  - 111 is negative, so not largest positive
- smallest negative number: 100 = -4

Humna Sultan

10/24/22 + 10/26/22

Binary
4 bits
1011 + 0101 = 10000 → last 4 digits represent 0
1011 = -5
0101 = 5

- If you want to look at 1011 as positive, the answer is 2^n - negative value, n being the number of bits
    - EX: 2^n - 5 = 16 - 5 = 11 → representation of 1011 in positive

- ~ means bitwise inversion (NOT operator)

x = 0001
~x = 1110
-8 + 4 + 2 = -2

```
>>> x = 1
>>> ~x
-2
>>>
```

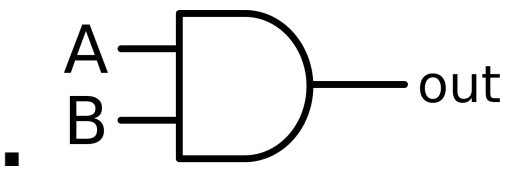
Programming + Bits

- ★ Floating point numbers are affected by the number of bits and the hardware used
    - 0.1 is not actually stored as 0.1, but rather 0.1000000000000xxxx
- ★ chr() gives you the ASCII symbol value for a number
- ★ ord() gives you the number that corresponds with the given ASCII value

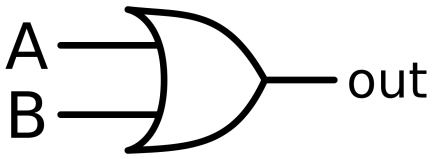
Adders + Gates

- Computing digitally
    - Adder
    - Multiplier
    - CPU

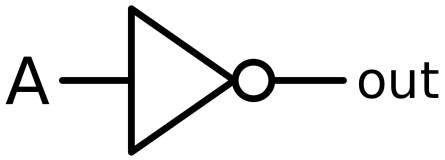
- Gates
    - XOR → only if one of the bits provided is a value
    - AND gate → 2 inputs, 1 output → x AND y is xy



    - OR gate → 2 inputs, 1 output → x OR y is x + y



- NOT gate → 1 input, 1 output (unary)



x	x_
0 (false)	1 (true)
1 (true)	0 (false)

★ 1-bit adder
- x_y + xy_
  - IN MY NOTES, _ is a negation on the previous character

- XOR gate → exclusive or, only returns 1 if both inputs are opposites

<u>Minterm Expansion Principle</u>

1. Write the truth table for the boolean function
2. Delete all the rows from truth table where the value of the function is 0
3. For each remaining row, we will create a “minterm”
   a. For each variable, if it’s value is 1, write the name of the variable as is
      i. If it’s value is 0, write the negated variable
   b. Now “AND” all of these variables
4. Combine all the minterms using “OR”

★ Available in textbook chapter 4.3.2

★ Full adder
- Input: x, y, carryIn
- Output: output, carryOut

Humna Sultan

10/28/22

Adding Circuits

sum = x XOR y XOR carry_in

x	y	carry_in	carry_out
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	1

- Use minterm expansion - keep only the values that have an output of 1

x_y cin
xy_cin
xycin_
Xycin

x_y cin + xy_cin + xycin_ + xycin
cin(x_y + xy_) + xy(cin_ + cin)

★ x_ + x is always equal to 1

cin(x_y + xy_) + xy(cin_ + cin)
xy(cin_ + cin) = 1

- cin(x XOR y) + xy → this is carry_out!!
- xy + x cin + y cin → another formula for carry_out (different solution) !!

Additional Notes

➔ For a NOT gate to produce a value in a relay, you need a value of 0 inputted
  ◆ The NOT of 0 is 1, and a 1 makes the circuit function
➔ For an AND gate to produce a value in a relay, you need both values inputted to be true (so the whole AND is true and electricity runs)

★ AND, OR, and NOT are known as the “universal set” in boolean algebra
  ○ Can be used to make any other boolean operators
★ DeMorgan’s Law:
  ○ x_y_ = x_ + y_
  ○ (x+y)_ = x_y_

De-Morgan's Second Theorem Verification						
A	B	$A + B$	$\overline{A + B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Memory

- NOR = OR output negated
- NOR gate with Set and Reset has brief moments of instability

★ SR-Latch: Latch with S and R
★ D-Latch: one input goes as is, the other is negated
  - Strobe and the negated input goes into R
  - Strobe and the input that is not negated (known as D) goes into S
  - Strobe = outlet, input only comes from D
★ D-latch makes SR-Latch safer and foolproof; memory will be stable
★ D-latches are used in RAM

10/31/22 (halloween hehe)

- Ian Turing - created Enigma (first modern computer)
- Joan Clarke - WW2 Code Breaker

S-R and D-Latches

- D-latches allow you to avoid S and R being 1 at the same time
    - You can only turn on either S or R at a time, cannot do both at once

RAM

- Parts
    - 2 address bits
    - 3 data bits (in)
    - read/write
    - 3 data bits (out) - on right side
- There is a connection between memory and CPU
- D-latches in RAM
    - One input in strobe is the address
    - The other input in strobe is read or write, depending on what is happening
    - Each of the 3 data bits is attached 1 to each column
    - Each row of Q's (3 columns) attaches to each of the output 3 data bits (1 column to each bit)
- Generally you have 8, 16, 32, 64 address bits

Calculator Register

Operation	Meaning
00	add
01	subtract
10	multiply
11	divide

Instruction / operation	destination	source	Second input
I0 and I1	D0 and D1	S0 and S1	T0 and T1

- 8 bits total

★ Encode the instruction register using the tables

EX:
Instruction register: Register2 = Register0 + Register2
Instruction register → 00100010
00 = operation (add)
10 = destination (2)
00 = source 1 (0)
10 = source 2 (2)

- Included in chapter 4.4.1

★ You increment the program counter when the operation has been fulfilled, then you get the next instruction in the instruction register
    - This is the connection between CPU and memory - CPU indicates that task has been fulfilled and gives next task

11/2/22

00 11 00 10

- First two bits mean instructions (add)

Register 3 = Register 0 + Register 2

HMMM

- Operations: add, sub, mul, div
- setn sets values
- addn adds to a register
- read = read from keyboard
- write = write to screen
- halt = stop
- nop = nothing

Area of a Triangle Program in HMMM

- Involves base, height; ½ * base * height = area

```
0 read r1  #height
1 read r2  #base
2 mul r1 r1 r2
3 setn r2 2  #to use the value 2 in division
4 div r1 r1 r2
5 write r1
6 halt
```

★ The program counter says what instructions to fetch from memory and is holding an address

"Feeling Jumpy" HMMM program - slide #7 in Lecture 8

My try (WRONG) - correct answer on next page:

```
0 read r1
1 read r2
2 setn r3 r2
3 jeqzn r3 8
4 mul r12 r1 r2
5 add r13 r12 r13
6 addn r3 -1
7 jumpn 3
8 write r13
9 halt
```

Humna Sultan

Correct answer

```
0 read r1  #assume is 5
1 read r2  #assume is 2
2 setn r13 1  #storing 1
3 jeqzn r2 7  #first iteration: 2 is not equal to 0 → keeps going
4 mul r13 r13 r1
5 addn r2 -1
6 jumpn 3
7 write r13
8 halt
```

Humna Sultan

11/4/22

Minterm Expansion

1. Keep rows that have an output of 1
2. Combine all variables in the rows with AND
    a. Values with a 0 are negated
3. Combine all rows with OR
4. Final answer → you can combine using formulas afterwards

Format for writing in python EX:

```
def f(x, y, z):
    return (y and z) or (x and not y and not z)
```

Two's Compliment

1. Solve for number in positive binary
2. Flip all values (0 to 1, 1 to 0)
3. Add 1 to the answer - done!

- In a programming language, you do not add 1 (causes overflow)

Smallest and largest numbers

- The leftmost digit is always negative
    - Largest positive cannot have a 1 in the leftmost spot
    - Smallest negative is equal to only a 1 in the leftmost
        - Putting 1's in the values towards the right will make the number more positive

QUESTION 3 IN PRACTICE

```
def listProd(L, M):
    if L == [ ]:
        return M
    elif M == [ ]:
        return L
    else:
        return [L[0] * M[0]] + listProd(L[1:], M[1:])
```

In Tail Recursion:

- Incorporate answer into function call

```
def listProd(L, M, R = [ ]):
        if L == [ ]:
                return R+M
        elif M == [ ]:
                return R+L
        else:
                return listProd(L[1:], M[1:], R + [L[0] * M[0]])
```

Humna Sultan

11/9/22

- Stacks
    - Last-In-First-Out
        - Last thing put in is the first you take out
    - First-In-Out-Last
        - First thing you put in is last thing you take out
    - Push and Pop
        - Push is put in, Pop is taking out

<u>"for" and "while" loop structures</u>
for variable in iterable:
    Body
    Body

- "For" and "in" are keywords
- "Variable" and "iterable" can be changed
    - "Iterable" can be a list, string, range, tuple, dictionary
        - You iterate and go to each element through them
- Only the indented lines after the for loop declaration are apart of the body

<u>Range</u>
range(10) = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- 10 numbers, but actual number 10 is not included (end value)
- range(start, end, step)

- Default start value for range = 0
- Default start value for step = 1

Humna Sultan

11/11/22

for <variable> in <iterable>
    Do stuff!

- Count-based loop
- For loop can go through iterables without viewing it as a list

while <condition>:
    Do stuff!

- Condition-based loop

```
>>> range(1, 10)
range(1, 10)
>>> print(range(1, 10))
range(1, 10)
>>> print(list(range(1,10)))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

x = 10
while x > 0:
    print(x)
    x = x + 1

- You must have the variable changing in the condition

★ In a flow chart of a loop, the condition is always in a diamond

List comprehension:
[print(n) for n in range(1,66,2)]

Humna Sultan

11/14/22

Factorial Function

```
def factorial(n):
    fact = 1
    for i in range(1, n+1):
        fact *= 1
    return fact
```

1st iteration: i = 1
fact = 1

2nd iteration: i = 2
fact = 2

3rd iteration: i = 3
fact = 6

4th iteration: i = 4
fact = 24

Functions

- .append() adds an element to the end of a list
- First lines of a function (“def —” + docstrings) are known as function signature

Arrays

A = [[1, 0, 1], [1, 1, 1], [0,1,0]]

- A [row] [col] = element

A [0] [3] is ERROR
A [1] [0] is 1

- Arrays in Python have pass by reference

```
>>> A = [1, 2, 3, 4]
>>> B = A
>>> B[0] = 42
>>> A[0]
42
>>>
```

B has the same reference as A

- You can use id() to see if two variables have the same reference

11/16/22

ASCII (explained in section 5.4 textbook)

- UTC - 8
- UTC - 16
- UTC - 32

Random Notes

- The value “x = 2020” cannot be represented in 8 bits (2020 is too big), so you need to use at least 16 bits
    - You can also use 32 or 64 bits
- Binary code - string of 7 symbols represents size

★ Methods - calls and references to variables
★ Heap - all of the values of objects
★ Garbage collector - removes memory references that are no longer being used

Code from class

```
>>> x = 2020
>>> y = 'A'
>>> z = 'vaccine'
>>> id(x)
1323911555920
>>> id(y)
1323877972592
>>> id(z)
1323912925424
>>> number = x
>>> id(number)
1323911555920
>>> # same id as x
>>> x = 42
>>> number
2020
>>> id(x)
1323871897104
>>> # x changed addresses
>>> x
42
>>> x + 1
43
>>> x = x + 1
>>> # the memory address will change again
>>> id(x)
1323871897136
>>> z = "vaccine"
>>> z[3] = "d"
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    z[3] = "d"
TypeError: 'str' object does not support item assignment
>>>
```

```
>>> z = "vaccine"
>>> z[3] = "d"
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    z[3] = "d"
TypeError: 'str' object does not support item assignment
>>> word = z
>>> id(z)
1323912925424
>>> id(wod)
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    id(wod)
NameError: name 'wod' is not defined. Did you mean: 'word'?
>>> id(word)
1323912925424
>>> # z and word have the same address
>>> z = "test"
>>> word
'vaccine'
>>> w = [1,2,3]
>>> z = w
>>> w[2] = 42
>>> z[2]
42
>>>
```

Humna Sultan

11/18/22

File I/O

- input()
    - Has one string parameter
    - EX: age = input(“Please enter your age: “)
        - Case as integer : age = int(input(“Please enter your age: “))
- open()
    - Parameters: open(fileName, mode)
    - Mode values:
        - “r” = reading
            - read()
            - readlines()
        - “w” = writing
            - write()
        - “a” = appending
    - If you open a file for writing and the file already exists / content already exists, all the previous content will be deleted and only the new stuff you are writing will be kept
    - If you open a file for appending and the file already exists / content already exists, the new information will be added onto the previous information in the file
- close()
    - NEEDED for when you open something
- KEYWORD: with
- strip(): remove spaces at the beginning and at the end of the string
- split(): splits the string at a specified separator, and returns a list
- join(): converts the elements of an iterable into a string
- capitalize(): converts the first character to upper case

- “\n” = new line character
    - Goes to the next line

Sorting

- sorted(____)
    - _____ = a list
    - You provide a list as a parameter and the original list provided as a parameter is not changed
        - You can set a variable equal to sorted()
    - Allows you to have an original list and an altered list
- listsort()
    - Does a sort “in place” - not making a copy of the list, but changing the actual list itself
        - Irreversible
- Both listsort() and sorted() use TimSort

- Combination of insertion sort and merge sort
- Runtime: nlogn

Humna Sultan

11/21/22

Sorting

- Functions sort() and sorted()
    - sorted() makes and returns a copy of the parameter, sort() does not
        - Both use TimSort - combo of merge sort and insertion sort
- Selection Sort - O(n^2)
    - Goes from starting index to the end of the list
    - Updates min index with the smaller value
        - Swaps elements - focuses on two at once and swaps both based on which is the min
- Quick Sort - O(nlogn)
    - Recursive sorting algorithm
    - Compares values of i and j (indexes) to swap values
        - changes i by 1 until i <= j
- Big O Notation: Describes runtime - runtime analysis

- Pseudocode: technique used to describe the distinct steps if an algorithm in a manner that is easy to understand from anyone with basic programming knowledge
    - Better readability, ease up code construction, act as a start point for documentation, easier bug detection and fixing

11/28/22

- Pseudocode - writing things without language specific syntax
    - Not language specific
    - Better readability
- Pseudocode constructs
    - Sequence
    - While
    - Repeat until (NOT USED IN PYTHON)
    - For
    - Case
    - If-then-else
- Make sure to follow pseudocode rules (in pseudocodes slideshow)
    - Most difficult: keep your statements programming language INDEPENDENT ; use the naming domain of the problem
        - EX: “append the last name to the first name” instead of “name = first + last”

11/30/22 + 12/2/22

Object Oriented Programming

- Consists of classes and constructors
- Object belongs to a class

- References can make a big difference

Overloaded Operator Naming

- Allows you to use operators on objects without comparing the ids - compares attributes of objects instead

Loop Tracing

for i in range(1, n+1):

- i is variable that is incremented
    - End value of range is excluded

Humna Sultan

12/5/22

Try and Except in Python

try:
—------- #possible error generating code here
except <optional error name afterwards>:
—------ #handle the error exception
else: (optional!)
—--------- #this part will run when no exceptions caught
finally: (optional!)
—---------- #whether you run except or not, this part will be executed

- ★ Things defined in try are not accessible outside (scope is limited)
    - Know valueError, nameError, genericError, fileNotFound, zeroError
- ★ You can have several excepts in one sequence
- ★ If an exception is found by the try and that exception does not meet any of the specific except statements, the generic exception runs (the one that does not have an optional error name afterwards)
- ★ Put the generic exception AFTER all of the specific exceptions (otherwise, the generic exception will run every time)

Additional Notes

- ★ Type Casting - int(), str(), etc.

12/9/22 + 12/12/22

Inheritance Stuff

- Relationship for inheritance: “is a”
    - Student “is” a undergrad student
    - Student “is” a grad student
        - Student is super class
        - Undergrad and grad are subclasses
- “Car” is a subclass of “vehicle”

- ★ “object” is the overarching parent class over every class in python