Lesson 2

Number System I . Selection . Iteration

Binary → Denary

- Base 2 to Base 10
- Multiply by place value

Place value	256	128	64	32	16	8	4	2	1
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Binary digit				1	1	1	0	0	1

$$(1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$
$$= 32 + 16 + 8 + 0 + 0 + 1 = 57$$

Denary → Binary

- Base 10 to Base 2
- Division by 2 method

In this method, we divide the denary number repeatedly by 2. The remainders of each division are then used to derive the binary number. The general algorithm is as follows:

Step 1: Draw a table with three columns – one column for denary numbers, one column for the quotients and one column for the remainders.

Step 2: Fill in the denary number in the first row.

Step 3: Divide the denary number by 2 and fill in its quotient and remainder in the same row.

Step 4: If the quotient is 0, proceed to Step 5. Otherwise, copy the quotient to the denary number column of the next row and repeat Step 3.

Step 5: The equivalent binary number is the remainder column read from the bottom up.

Try now on a piece of paper (7 mins)

- 101010111 to denary
- 6543 to binary

print() VS return

- Once you print, cannot use
- output (return) of a function can be used as an input of another function

input()

The `input()` function accepts a `str` to use as a prompt and enables the user to key in the input. The input ends once the user presses the Enter key, and everything the user has typed (not including the prompt) will be returned from the `input()` function call as a `str`.

```
>>> name = input("Enter age: ")
Enter age: 15
>>> print(int(age))
15
```

Useful operators

Operator	Example	Equivalent
+=	x += a	x = x + a
-=	x -= a	x = x - a
*=	x *= a	x = x * a
/=	x /= a	x = x / a
//=	x //= a	x = x // a
**=	x **= a	x = x ** a
%=	x %= a	x = x % a

in , not , and , or

Operator	Name	Description	Examples

`in`	Membership	Returns the `bool` value `True` if the value on the left can be found inside the sequence on the right and `False` if it cannot be found	`>>> "C" in "Computing"`
			`True`
			`>>> "c" in "Computing"`
			`False`

Operator	Name	Description	Examples

`not`	Negation	Takes a `bool` and returns its opposite value	`>>> not True`
			`False`
			`>>> not False`
			`True`

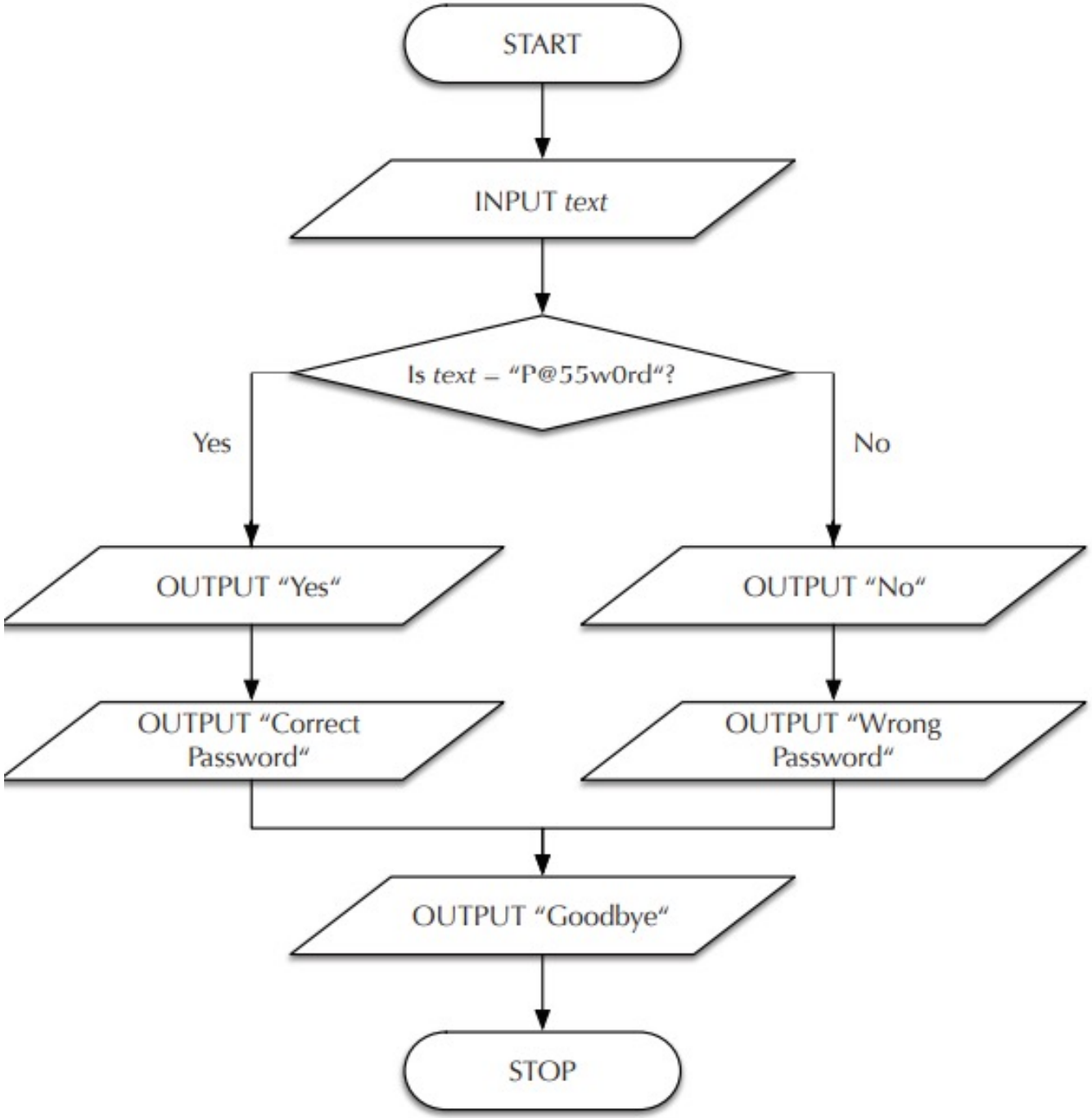
`and`	Conjunction	Returns the `bool` value `True` if both values are `True` and `False` if either one or both values are `False`	`>>> True and True`
			`True`
			`>>> True and False`
			`False`

`or`	Disjunction	Returns the `bool` value `True` if either one or both values are `True` and `False` if both values are `False`	`>>> True or True`
			`True`
			`>>> True or False`
			`True`

Selection

if – elif – else statements:

Indentation is important




```
text = input("Enter text: ")
if text == "P@55w0rd":
    print("Yes")
    print("Correct Password")
else:
    print("No")
    print("Wrong Password")
print("Goodbye")
```

Why different indentation ?

We might want to provide a separate error message if the text entered is blank.

```
text = input("Enter text: ")
if text == "":
    print("Blank Input Not Allowed")
elif text == "P@55w0rd":
    print("Yes")
    print("Correct Password")
else:
    print("No")
    print("Wrong Password")
print("Goodbye")
```

Even / odd ? (demonstrate)

- Using input() or define a function

Iteration – for loop

Syntax 4.5 for-in Statement

```
for name in sequence:
    commands to repeat for each item in the sequence
```

```
for i in range(5):
    print(4)
```

```
for i in range(5):
    print(i)
```

```
for char in 'abcdef':
    print(char)
```

Iteration – while loop

Syntax 4.4 `while` Statement

```
while condition:
    commands to repeat while condition is True
```

for loop vs while loop (demonstrate)

Average example: find the average of 5 numbers

Max, min,

Problem Solving - Man, Cabbage, Goat, Wolf

A man lives on the east side of a river. He wishes to bring a cabbage, a goat and a wolf to a village on the west side of the river to sell. However, his boat is only big enough to hold himself, and either the cabbage, goat or wolf. In addition, the man cannot leave the goat alone with th
the cabbage, and he canno
because the wolf will eat th
problem ?

Abstraction – a representation that leaves out details of what is represented

Colour of boat relevant? Width of river? Name of man?

Start state: man cabbage goat wolf

[E , E , E , E]

In this representation, the symbol E denotes that each corresponding object is on the east side of the river.

If the man were to row the goat across with him, the representation of the new state:

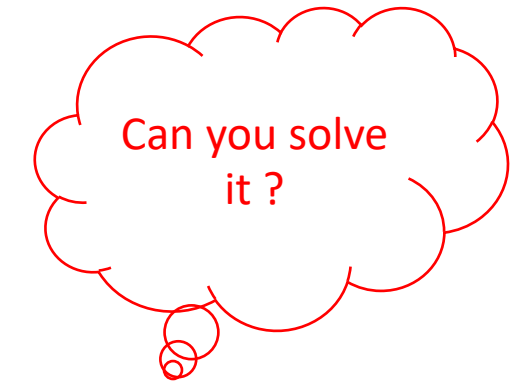
New state: man cabbage goat wolf

[W , E , W , E]

… … …

Goal state: man cabbage goat wolf

[W , W , W W]

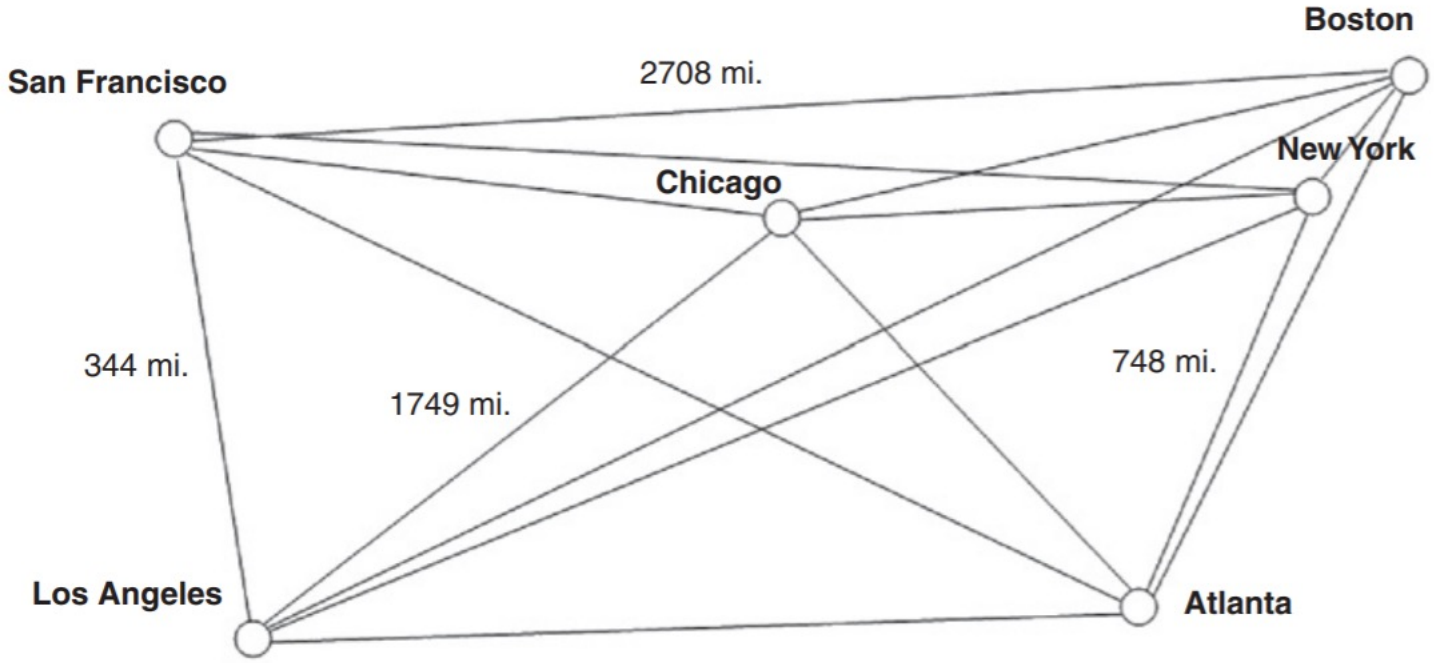


In order to computationally solve a problem, two things are needed:

a **representation** of the problem and an **algorithm** that solves it.

Limits of computational problem solving

- Travelling salesman Problem
  - The problem is to find the shortest route of travel for a salesman needing to visit a given set of cities.



Travelling salesman Problem

- Using a brute force approach, the lengths of all possible routes would be calculated and compared to find the shortest one.
- For 3 cities, the number of possible routes is 3! = 1 x 2 x 3 = 6
- For 10 cities, the number of possible routes is 10! = 1 x 2 x … x 10 = 3628800
- For 20 cities, the number of possible routes is 20!
- If we assume that a computer could compute the lengths of one million routes per second, ...
- For 50 cities …

Any algorithm that correctly solves a given problem must solve the problem in a reasonable amount of time, otherwise it is of limited practical use.

An algorithm is a finite number of clearly described, unambiguous “doable” steps that can be systematically followed to produce a desired result for given input in a finite amount of time.

Algorithms and Computers: A Perfect Match

- Because computers can execute instructions very quickly and reliably without error, algorithms and computers are a perfect match.
- Example: determining the day of the week for any date between January 1, 1800 and December 31, 2099.

To determine the day of the week for a given **month**, **day**, and **year**:

1. Let **century_digits** be equal to the first two digits of the year.
2. Let **year_digits** be equal to the last two digits of the year.
3. Let **value** be equal to **year_digits** + floor(**year_digits** / 4)
4. If **century_digits** equals 18, then add 2 to **value**, else
   if **century_digits** equals 20, then add 6 to **value**.
5. If the **month** is equal to January and **year** is not a leap year,
   then add 1 to **value**, else,

   if the **month** is equal to February and the **year** is a leap year, then
   add 3 to **value**; if not a leap year, then add 4 to **value,** else,

   if the **month** is equal to March or November, then add 4 to **value**, else,

   if the **month** is equal to May, then add 2 to **value**, else,

   if the **month** is equal to June, then add 5 to **value**, else,

   if the **month** is equal to August, then add 3 to **value**, else,

   if the **month** is equal to October, then add 1 to **value**, else,

   if the **month** is equal to September or December, then add 6 to **value**,
6. Set **value** equal to (**value** + **day**) mod 7.
7. If **value** is equal to 1, then the day of the week is Sunday; else

   if **value** is equal to 2, day of the week is Monday; else

   if **value** is equal to 3, day of the week is Tuesday; else

   if **value** is equal to 4, day of the week is Wednesday; else

   if **value** is equal to 5, day of the week is Thursday; else

   if **value** is equal to 6, day of the week is Friday; else

   if **value** is equal to 0, day of the week is Saturday

Given that the year 2016 is a leap year, what day of the week does April 15th of that year fall on?

Possible to write it as a function in Python?