Python Basics

Lists, range and loop

CS101 Lecture #8

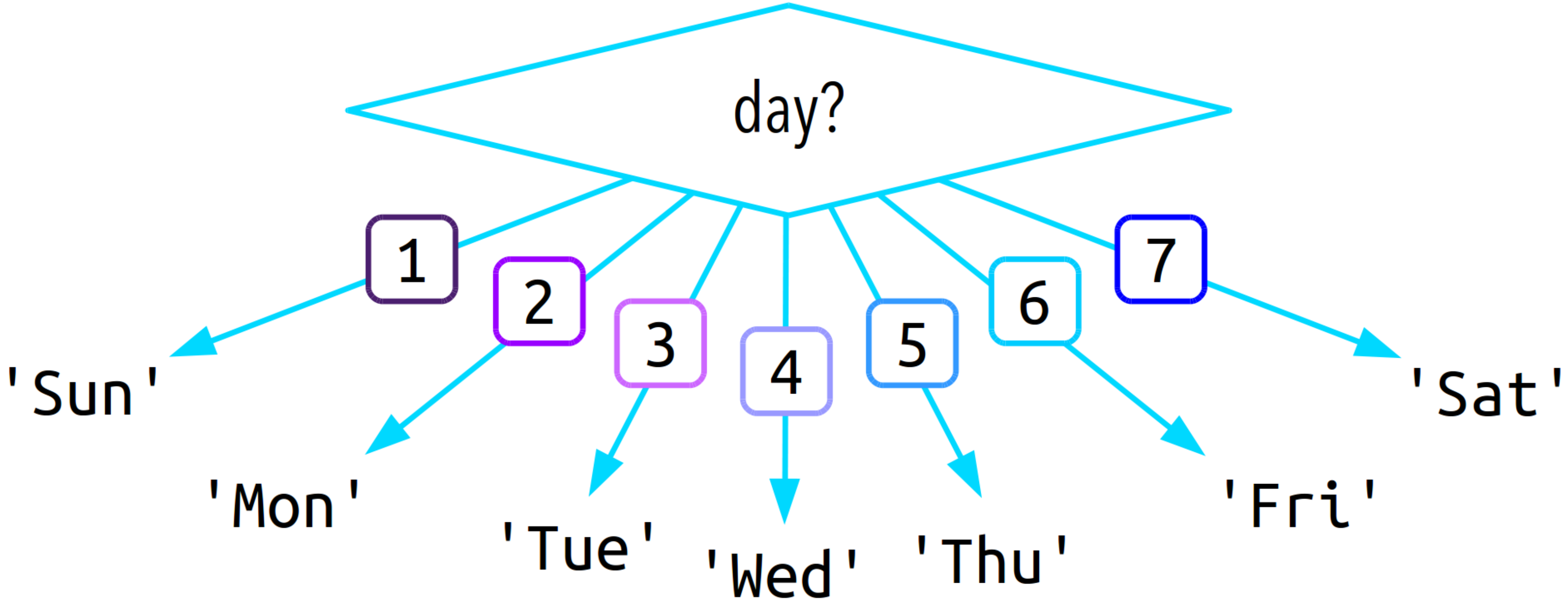
2016-10-19

Administrivia

Administrivia

- Homework #4 is out
- Due day is Oct 23, 6pm.

Warmup: `if-elif-else`


day?
1
2
3
4
5
6
7
'Sun'
'Mon'
'Tue'
'Wed'
'Thu'
'Fri'
'Sat'

```
if day == 1:
    print(‘Sunday’)
if day == 2:
    print(‘Monday’)
if day == 3:
    print(‘Tuesday’)
if day == 4:
    print(’Wednesday’)
if day == 5:
    print(‘Thursday’)
if day == ‘6’
    print(‘Friday’)
if day == ‘7’
    print(‘Saturday’)
else:
    print(‘Not a valid input’)
```

```
if day == 1:
    print(‘Sunday’)
else:
    if day == 2:
        print(‘Monday’)
    else:
        if day == 3:
            print(‘Tuesday’)
        else:
            if day == 4:
                print(’Wednesday’)
            else:
                if day == 5:
                    print(‘Thursday’)
                else:
                    if day == ‘6’
                        print(‘Friday’)
                    else:
                        if day == ‘7’
                            print(‘Saturday’)
                        else:
                            print(‘Not a valid input’)
```

```
if day == 1:
    print(‘Sunday’)
elif day == 2:
    print(‘Monday’)
elif day == 3:
    print(‘Tuesday’)
elif day == 4:
    print(’Wednesday’)
elif day == 5:
    print(‘Thursday’)
elif day == ‘6’
    print(‘Friday’)
elif day == ‘7’
    print(‘Saturday’)
else:
    print(‘Not a valid input’)
```

Warmup: *Recursion*

Question

Every problem can be formulated in the form of *recursion*?

A. Yes
B. No

Question

Only a set of special problems can be formulated by *recursion*

A. Yes
B. No

Question

If a problem can be formulated as recursion, then *recursion* is the best way to solve it

A. Yes
B. No

Question

Recursive algorithms are also called ______ (sometimes) in computer algorithm design?

A. Dynamic programming
B. Divide and conquer
C. Brute force
D. Randomization

Container Data Type

Container

Example

```
colors = [‘red’, ‘green’, ‘blue’, ‘cyan’, ‘megenta’, ‘yellow’]
```

`list` *data type*

- It represents an listed collection of items
- It is a `container` data type
- Also an `iterable` data type
- Can hold values of any type, and they don't have to be the same (not the same as `array`)

How to create a `list`?

- Syntax:
  - An opening bracket [
  - One or more comma-separated data values
  - A closing bracket]

```
x = [10, 3.14, ‘2.71’]
```

How to access a list

- Works a bit like strings:

```
x = [10, 3.14, ‘2.71’]

print(x[0])
print(x[1:3])
print(len(x))
```

Modify a list?

- Modify a string
  - Strings are ***immutable***; we cannot change its content without creating a new string

```
s = ‘strang’
s[3] = ‘i’                    #nope

s = s[:3] + ‘i’ + s[4:]    #correct
```

Modify a list?

- We can change content of a list – it's ***mutable***

```
x = [4,1,2,4]
x[3] = -2          #item assignment
x.append(5)        #add an item to the end
x.sort()           #sort items by value
del x[1]           #delete an item
```

for-loop

- How to iterate a list?
  - print out all items of a list

```
colors = [‘red’, ‘yellow’, ‘blue’, ‘purple’, ‘jale’]
```

Loops

for loop

```
colors = [‘red’, ‘green’, ‘blue’, ‘cyan’, ‘megenta’, ‘yellow’]

for color in colors:
    print(color)
```

for loop

- A for loop requires:
  - Keyword `for`
  - A loop variable
  - Keyword `in`
  - An iterable data type
  - A **block** of code

- For can iterate over items of a *iterable* type one at a time

Example

```
s = ‘abcdefg’
t = ‘’

for c in s:
   t = c + t
```

What’s the value of t?

- A ‘abcdefg’
- B ‘gfedcba’
- C ‘a’
- D ‘g’

Exercise

Write a function to sum up all digits in a number, i.e.,

12145 → 1 + 2 + 1 + 4 + 5 → 13

Solution

```
def sum_digits(n):
  result = 0
  for digit in str(n):
    result = result + int(digit)
  return result
```

Exercise

Write a function to sum up numbers from 1 to 100

Solution

```
results = 0
for i in range(1, 101):
   result = result + i
```

`range` function

- The range function returns an ***iterator*** containing integers in a specified range
- range can be casted as a list
  - list(range(1, 10))
- Two arguments:
  - (optional) the starting value (inclusive)
  - The ending value (exclusive)

while-loop

Write a function to sum up numbers from 1 to 100

```
result = 0
i = 1
while i <= 100:
   result = result + i
   i += 1
```

while-loop

- A `while` loop has:
  - The keyword `while`
  - A logical comparison (bool-valued result)
  - A **block** of code

Example

```
x = 3
While x > 0:
   print(‘Hello’)
   x -= 1
```

How many times is ‘Hello’ printed?

A 0

B 1

C 2

D 3

Example

```
i = 0
count = 0
while i < 100:
   if (i%2) == 1:
      count += 1
   i += 1
```

What is this piece of code doing?

while-loop

Write a program for a user to create a new password. The program should accept a password attempt `pwd` from the user and check it with the function *`validate_password(pwd)`*. If the password is valid, the program ends. If the password is invalid, the program asks for a new attempt, repeating until the user enters a valid password.

Solution

```
pwd = input(‘Enter a password: ’)

while not validate_password(pwd):
  pwd = input(‘INVALID! Try again: ’)

print(‘Password correct!’)
```

Infinite loop

- Make sure your code always has a way to end

```
While True:
   print(‘Hello!’)
```

Infinite loop

- Make sure your code always has a way to end

```
While True:
   print('Hello!')
```

- Use 'Ctrl+C' to force break

Infinite loop

- Make sure your code always has a way to end
- Use `break`

```
x = 3
while True:
   print('Hello!')
   x -= 1
   if x == 0:
      break
```

Accumulator pattern

- *Design patterns* are common structures we encounter in writing code
- The *accumulator* pattern uses an accumulator variable to track the progress of a loop

```
i = 0
sum = 0
while i <= 4:
    sum += i
    i += 1
```

Exercise

Write a function to sum up all digits in a number, i.e.,

12145 → 1 + 2 + 1 + 4 + 5 → 13

(use `while` loop)

Solution

```
def sum_digits(n):
    …
```