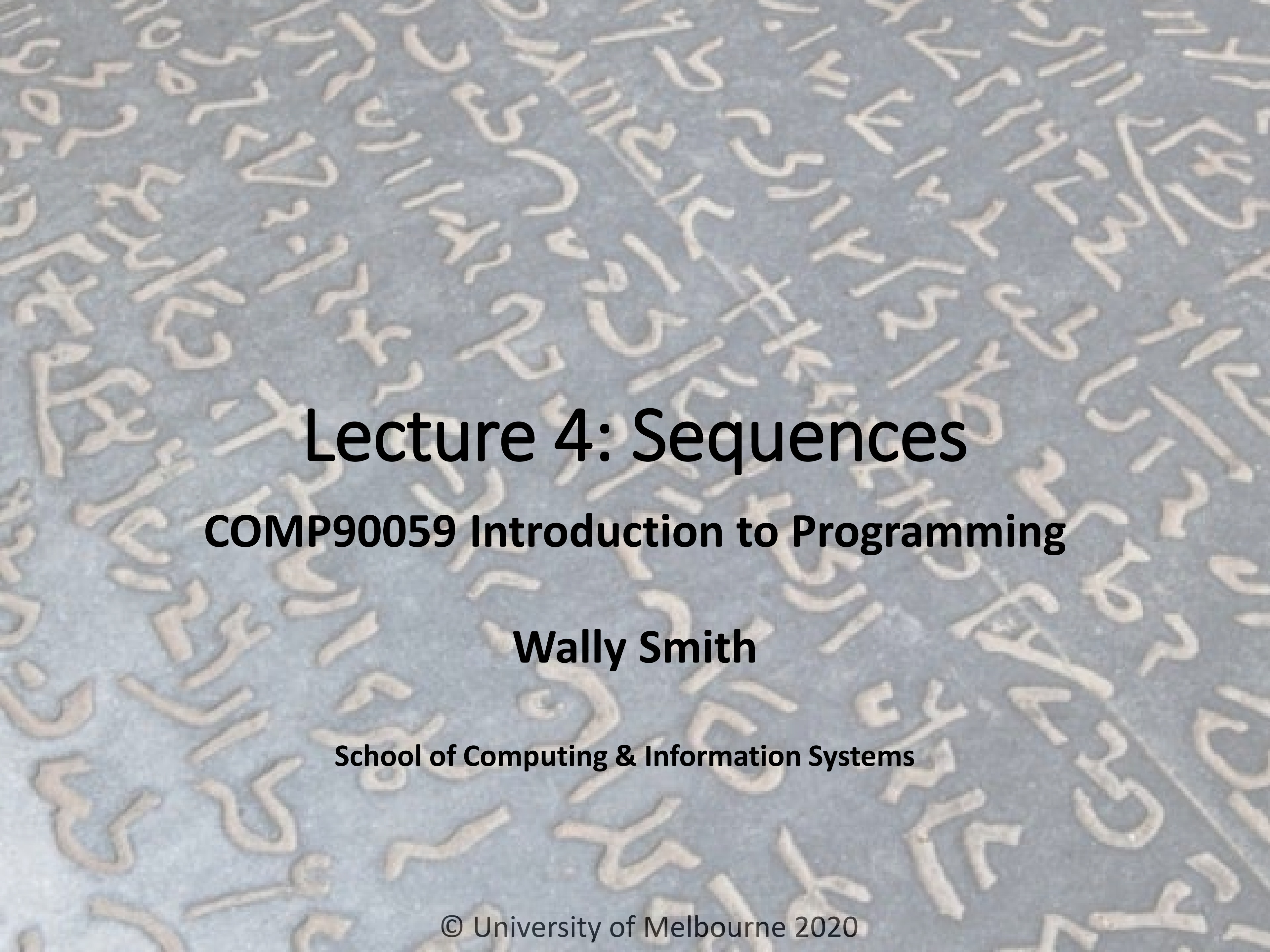


Lecture 4: Sequences

COMP90059 Introduction to Programming

Wally Smith

School of Computing & Information Systems

Lecture Overview

- Reviewing the Challenges set from last week: perfecting the calculation in the daysOfLife program; designing the tennis scoring program
- Sequences - data made up of an ordered set of elements
  - strings
  - lists
- String operations
- Lists & list operations
- Methods for strings and lists
- String formatting
- New challenges:
  - improving the elegance of the daysOfLife program
  - developing the passwordSecurity program

Reviewing Exercises from Lecture 3 Conditionals

```
#program to tell which of three int numbers is biggest
num1 = int(input('Enter number 1: '))
num2 = int(input('Enter number 2: '))
num3 = int(input('Enter number 3: '))

if num1 > num2:
           if num1 > num3:
               biggest = num1
           else:
               if num3 > num2:
                   biggest = num3
               else:
                    biggest = num2
else:
    if num2 > num3:
        biggest = num2
    else:
        if num3 > num1:
            biggest = num3
        else:
            biggest = num1

print('Biggest: ',biggest)
```

biggest = num3

biggest = num3

How to make this code more elegant?

```
#program to tell which of three int numbers is biggest
num1 = int(input('Enter number 1: '))
num2 = int(input('Enter number 2: '))
num3 = int(input('Enter number 3: '))

if num1 > num2:
    if num1 > num3:
        biggest = num1
      else:
        biggest = num3
else:
  if num2 > num3:
    biggest = num2
  else:
    biggest = num3

print('Biggest: ',biggest)
```

How to make this code more elegant?

```
if num1 > num2  and num1 > num3:
        biggest = num1
elif num2 > num1 and num2 > num3:
        biggest = num2
else:
        biggest = num3
```

Better because it's READABLE
(i.e. a person can easily understand it)

```
biggest = num1
if num2 > biggest:
        biggest = num2
if num3 > biggest:
        biggest = num3
```

Even better because it's not only VERY READABLE,
but also EXTENSIBLE to more than three numbers

MORAL: There are many correct solutions to a coding problem.
Some are ELEGANT, many are not.

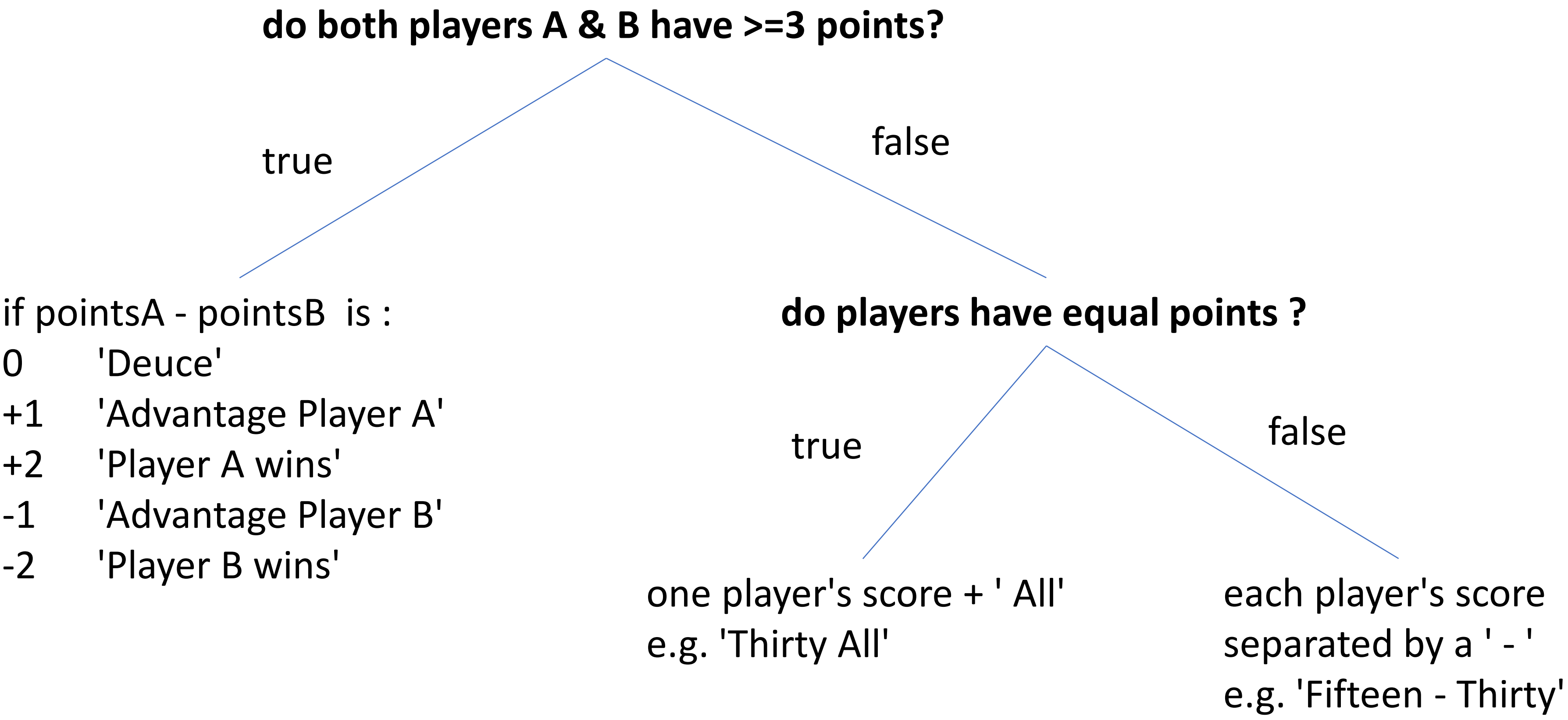
ELEGANT CODE is efficient, while still being readable, and if possible is extensible to more complex cases.

L3 Ex10: Scoring a game of tennis. Part 2

Write a program that takes in the number of points that two tennis players have scored against each other, and gives the game score so far.

Player A	Player B	Score
0	0	Love All
1	0	Fifteen - Love
1	2	Fifteen - Thirty
2	2	Thirty All
2	3	Thirty - Forty
2	4	Player B wins
3	3	Deuce
3	4	Advantage Player B
4	4	Deuce
4	5	Advantage Player B
6	5	Advantage Player A
7	5	Player A wins
9	11	Player B wins

Analysis of tennis scoring problem



do both players A & B have >=3 points?

true

false

if pointsA - pointsB is :

0	'Deuce'
+1	'Advantage Player A'
+2	'Player A wins'
-1	'Advantage Player B'
-2	'Player B wins'

do players have equal points ?

true

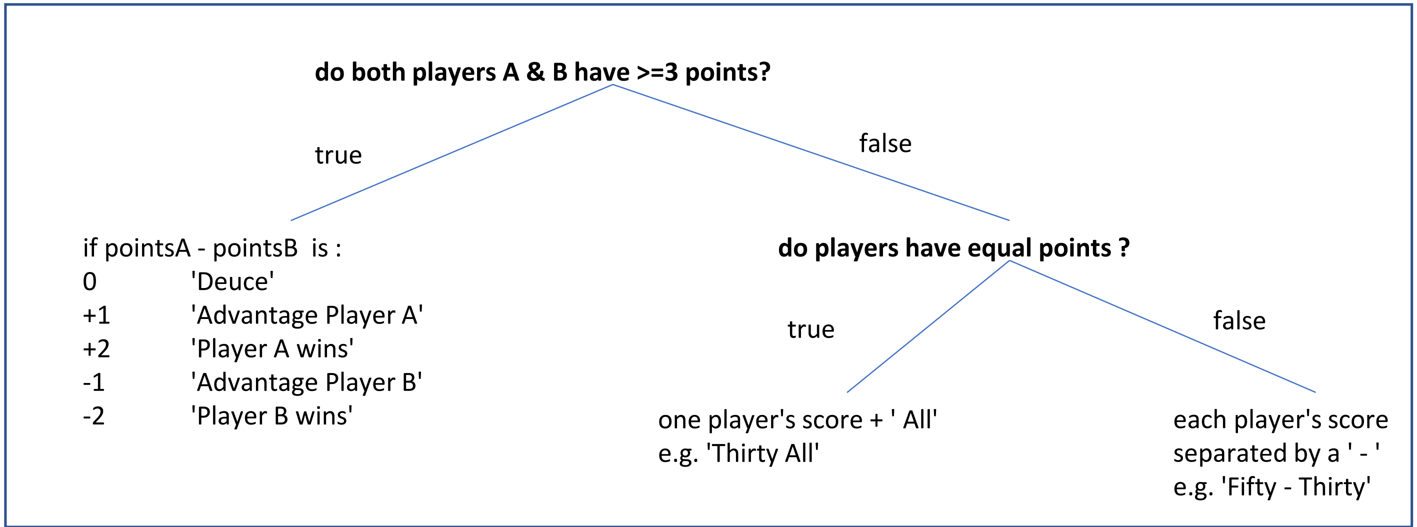
false

one player's score + ' All'
e.g. 'Thirty All'

each player's score
separated by a ' - '
e.g. 'Fifteen - Thirty'

Program design for the tennis scoring problem

```
#program to calculate tennis score V1

# take from user the number of points scored by each player
# calculate points difference between players for later decisions
# produce message for scores of deuce and above
# produce message for scores below deuce
        # when points are equal
        # when points are not equal
# report  the score to user
```



do both players A & B have >=3 points?

true

if pointsA - pointsB is :

0	'Deuce'
+1	'Advantage Player A'
+2	'Player A wins'
-1	'Advantage Player B'
-2	'Player B wins'

false

do players have equal points ?

true

one player's score + ' All'
e.g. 'Thirty All'

false

each player's score
separated by a ' - '
e.g. 'Fifty - Thirty'

L03 Ex 11: Perfecting the DaysOfLife program

Make the DaysOfLife program code perfectly accurate, taking into account the exact number of days in each months, and taking into account leap years.

To do this use ..

- if, elif, else statements
- the leap year calculation we developed in this lecture

Tip: Look at each point in the current DaysOfLife code where an approximation is used. Think how you could precede that part of the code with a series of if ... elif ... else statements that reset the approximate value to the correct value each time the calculation is made.

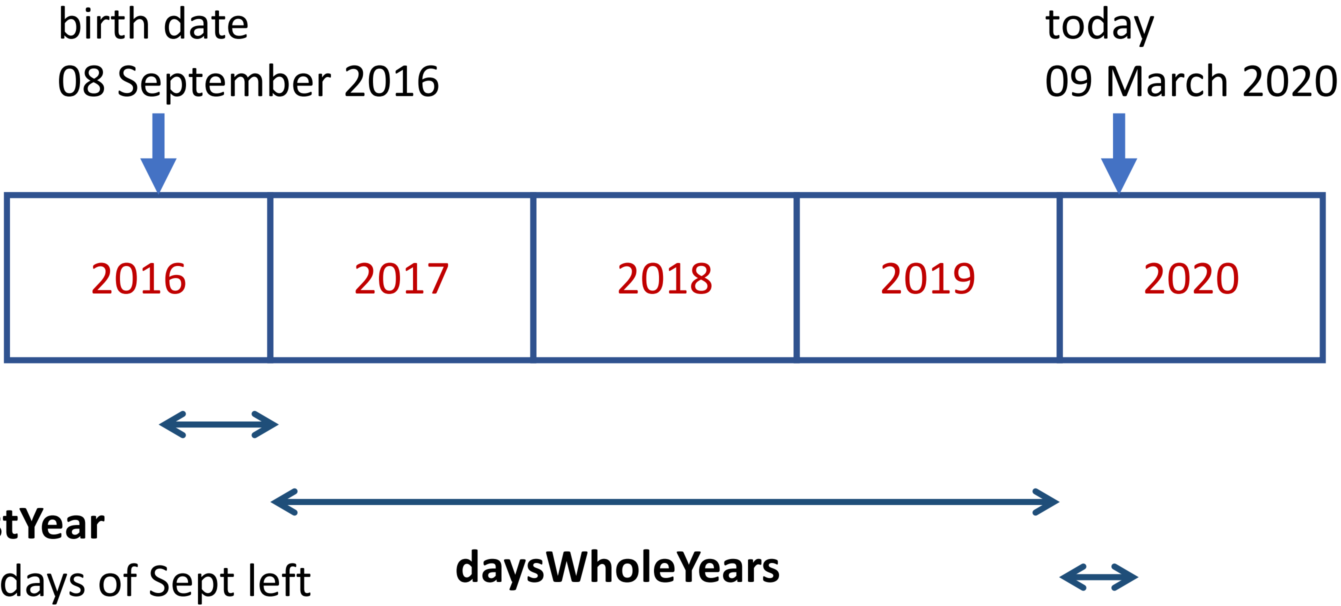
This requires several lines of extra code to be added to specify the correct situation for each month, or for each year under consideration. My solution adds about 50 extra lines of code.

In later lectures we will learn how to solve this in a more elegant way!

```
# Get birthdate from user as 3 values: year, month & day

dobYear = int(input('Enter year of birth :'))

dobMonth = int(input('Enter month of birth (1 -12) :'))

dobDay = int(input('Enter day of birth: '))

# Establish today's date as 3 values: year, month & day

todayYear, todayMonth, todayDay = 2020, 3, 9

# Work out days in incomplete first year of life (daysFirstYear)

daysFirstYear =  (12-dobMonth) * 30 + (30-dobDay)

# Work out how many  days in whole years of life (wholeYears)

daysWholeYears = (todayYear - dobYear - 1) * 365  # calculate without leap years

# Work out days in incomplete current year of life (daysCurrentYear)

daysCurrentYear = todayDay + (todayMonth-1)*30 #assume roughly 30 days per month

# Estimate number of leap years (leapYears)

leapyears = (todayYear - dobYear)//4

# Rough answer = daysFirstYear + daysWholeYears + daysCurrentYear + leapyears

estDaysOfLife = daysFirstYear + daysWholeYears + daysCurrentYear + leapyears

print('You have been alive very roughly', estDaysOfLife, 'days')
```

Analyzing the days of life calculation

Take someone who was born on 8 September 2016 ...



birth date
08 September 2016

today
09 March 2020

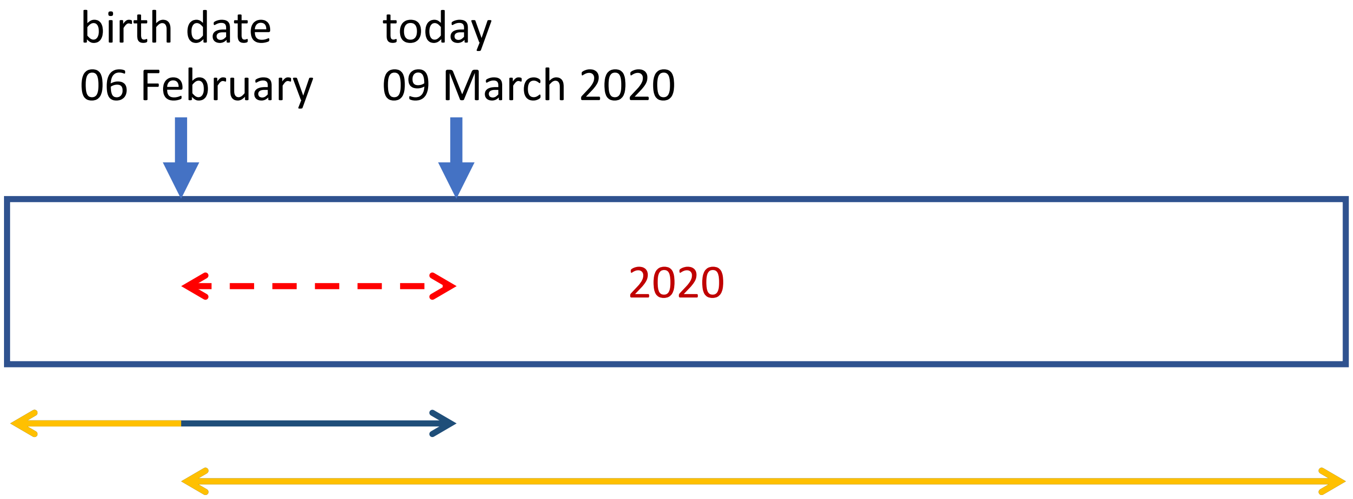
daysFirstYear
* 30 - 8 days of Sept left
* 12 - 9 = 3 months left
@ 30 days each

daysWholeYears
* years = 2020 - 2016 - 1
* @365 days per year

daysCurrentYear
* whole months = 3 - 1 @
30days each
* 9 days of March

What if birth date is this year?

Take someone who was born on 6 February 2020 ...



birth date
06 February

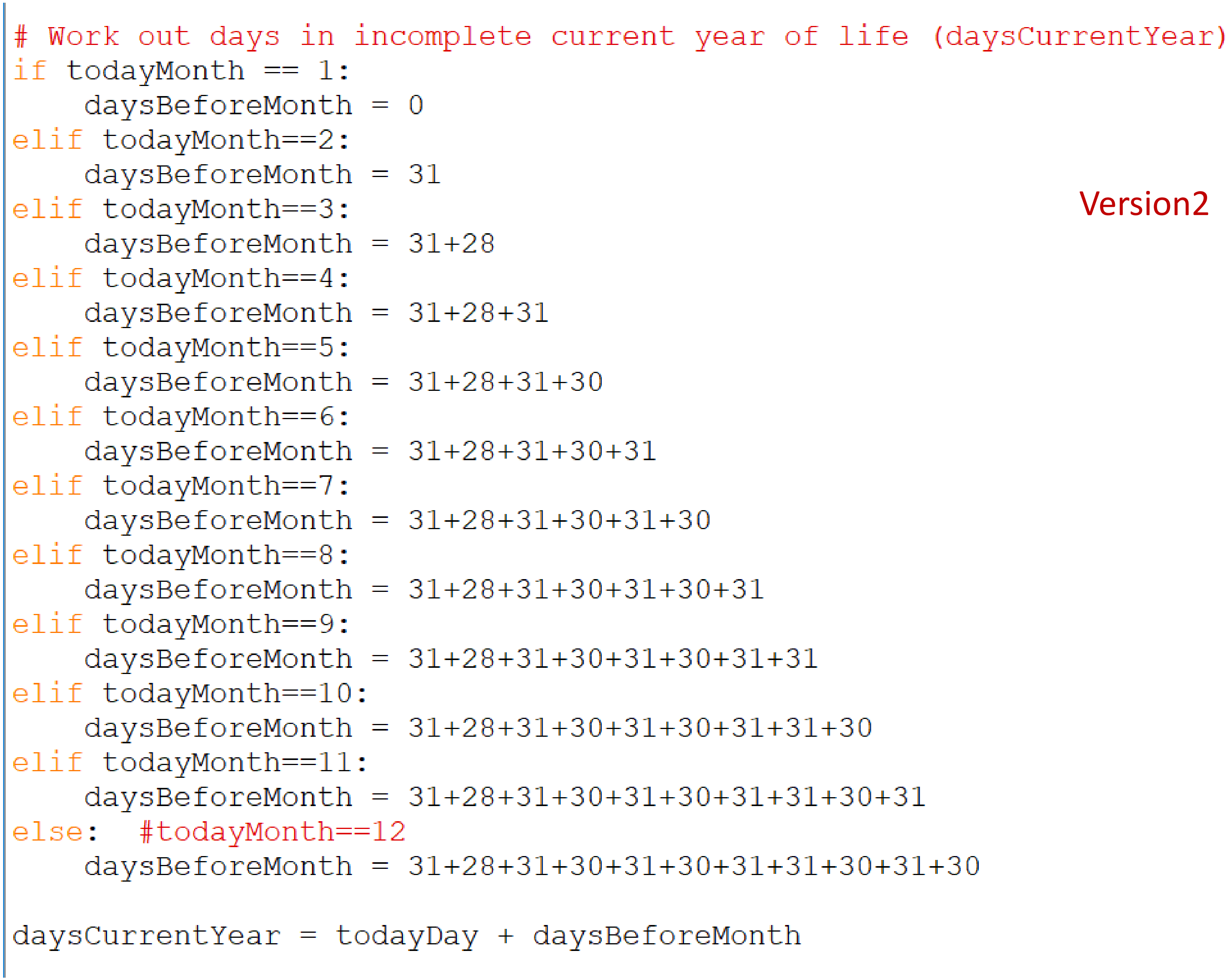
today
09 March 2020

2020

daysCurrentYear
* whole months = 3 - 1 @
30days each
* 9 days of March

daysWholeYears
* years = 2020 - 2020 - 1
@365 days per year = - 365

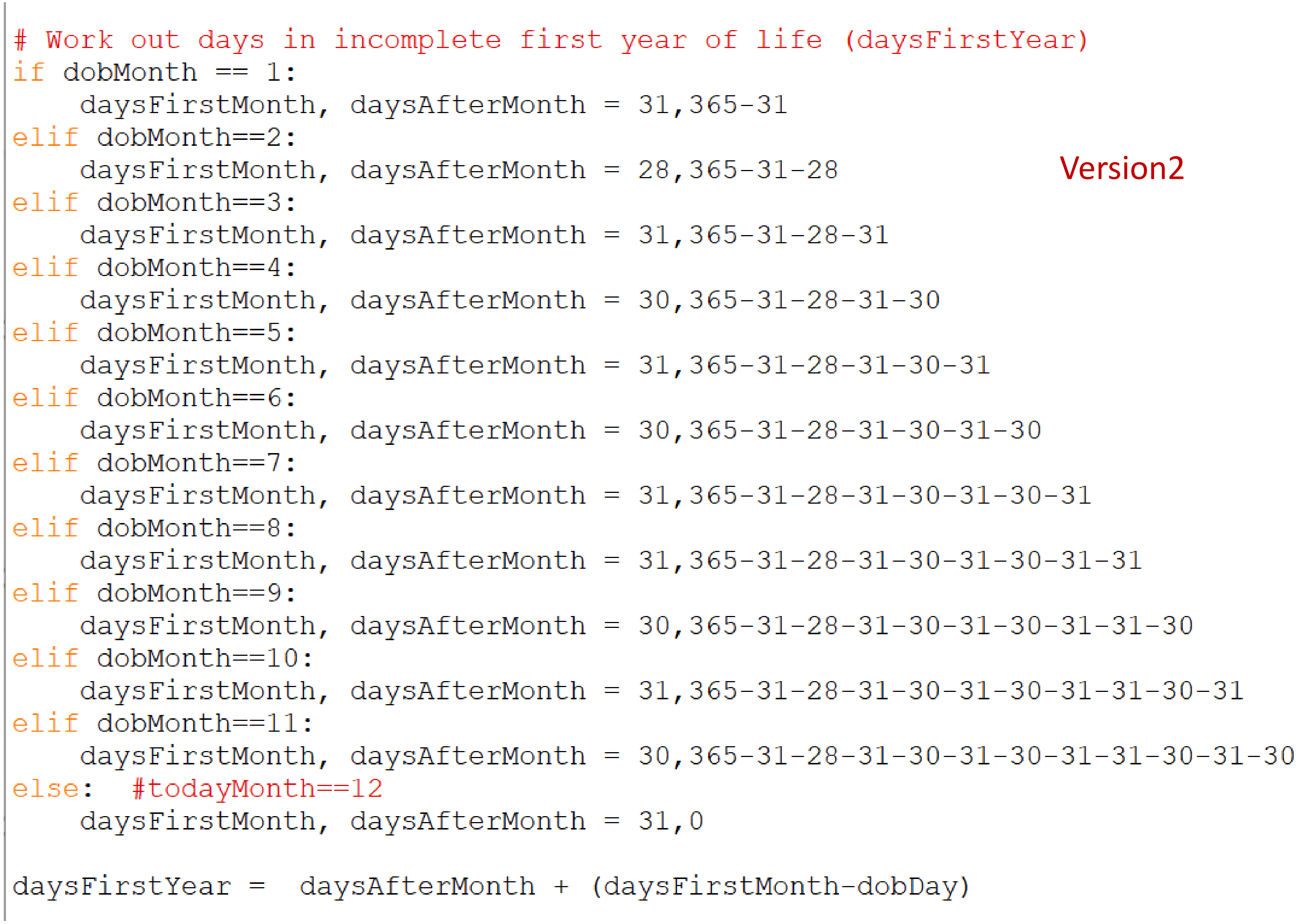
daysFirstYear
* 30 - 6 days of Feb left
* 12 - 2 = 10 months left
@ 30 days each



```
# Work out days in incomplete current year of life (daysCurrentYear)
if todayMonth == 1:
    daysBeforeMonth = 0
elif todayMonth==2:
    daysBeforeMonth = 31
elif todayMonth==3:
    daysBeforeMonth = 31+28
elif todayMonth==4:
    daysBeforeMonth = 31+28+31
elif todayMonth==5:
    daysBeforeMonth = 31+28+31+30
elif todayMonth==6:
    daysBeforeMonth = 31+28+31+30+31
elif todayMonth==7:
    daysBeforeMonth = 31+28+31+30+31+30
elif todayMonth==8:
    daysBeforeMonth = 31+28+31+30+31+30+31
elif todayMonth==9:
    daysBeforeMonth = 31+28+31+30+31+30+31+31
elif todayMonth==10:
    daysBeforeMonth = 31+28+31+30+31+30+31+31+30
elif todayMonth==11:
    daysBeforeMonth = 31+28+31+30+31+30+31+31+30+31
else:  #todayMonth==12
    daysBeforeMonth = 31+28+31+30+31+30+31+31+30+31+30

daysCurrentYear = todayDay + daysBeforeMonth
```

Version2

daysCurrentYear = todayDay + (todayMonth-1)*30 #assume roughly 30 days per month

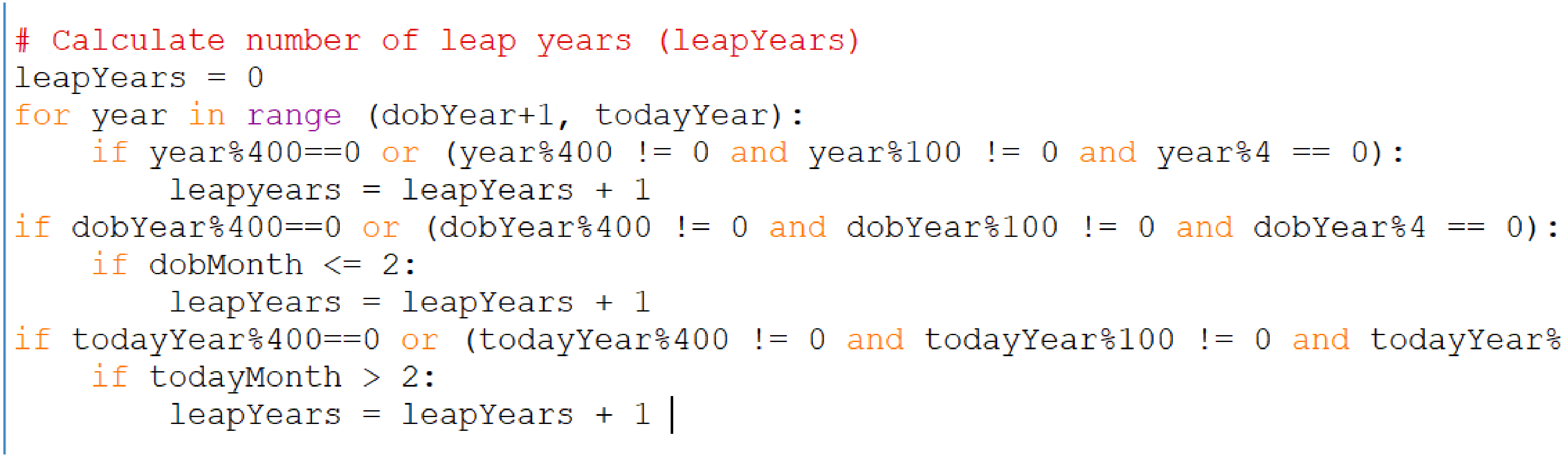


```
# Work out days in incomplete first year of life (daysFirstYear)
if dobMonth == 1:
    daysFirstMonth, daysAfterMonth = 31,365-31
elif dobMonth==2:
    daysFirstMonth, daysAfterMonth = 28,365-31-28
elif dobMonth==3:
    daysFirstMonth, daysAfterMonth = 31,365-31-28-31
elif dobMonth==4:
    daysFirstMonth, daysAfterMonth = 30,365-31-28-31-30
elif dobMonth==5:
    daysFirstMonth, daysAfterMonth = 31,365-31-28-31-30-31
elif dobMonth==6:
    daysFirstMonth, daysAfterMonth = 30,365-31-28-31-30-31-30
elif dobMonth==7:
    daysFirstMonth, daysAfterMonth = 31,365-31-28-31-30-31-30-31
elif dobMonth==8:
    daysFirstMonth, daysAfterMonth = 31,365-31-28-31-30-31-30-31-31
elif dobMonth==9:
    daysFirstMonth, daysAfterMonth = 30,365-31-28-31-30-31-30-31-31-30
elif dobMonth==10:
    daysFirstMonth, daysAfterMonth = 31,365-31-28-31-30-31-30-31-31-30-31
elif dobMonth==11:
    daysFirstMonth, daysAfterMonth = 30,365-31-28-31-30-31-30-31-31-30-31-30
else:  #todayMonth==12
    daysFirstMonth, daysAfterMonth = 31,0

daysFirstYear =  daysAfterMonth + (daysFirstMonth-dobDay)
```

Version2

daysFirstYear = (12-dobMonth) * 30 + (30-dobDay)

Version2



```
# Calculate number of leap years (leapYears)
leapYears = 0
for year in range (dobYear+1, todayYear):
    if year%400==0 or (year%400 != 0 and year%100 != 0 and year%4 == 0):
        leapyears = leapYears + 1
if dobYear%400==0 or (dobYear%400 != 0 and dobYear%100 != 0 and dobYear%4 == 0):
    if dobMonth <= 2:
        leapYears = leapYears + 1
if todayYear%400==0 or (todayYear%400 != 0 and todayYear%100 != 0 and todayYear%
    if todayMonth > 2:
        leapYears = leapYears + 1
```

leapyears = (todayYear - dobYear)//4

for ... in loops (an example of iteration)

iteration is a core technique in which a block of code is repeated in a loop

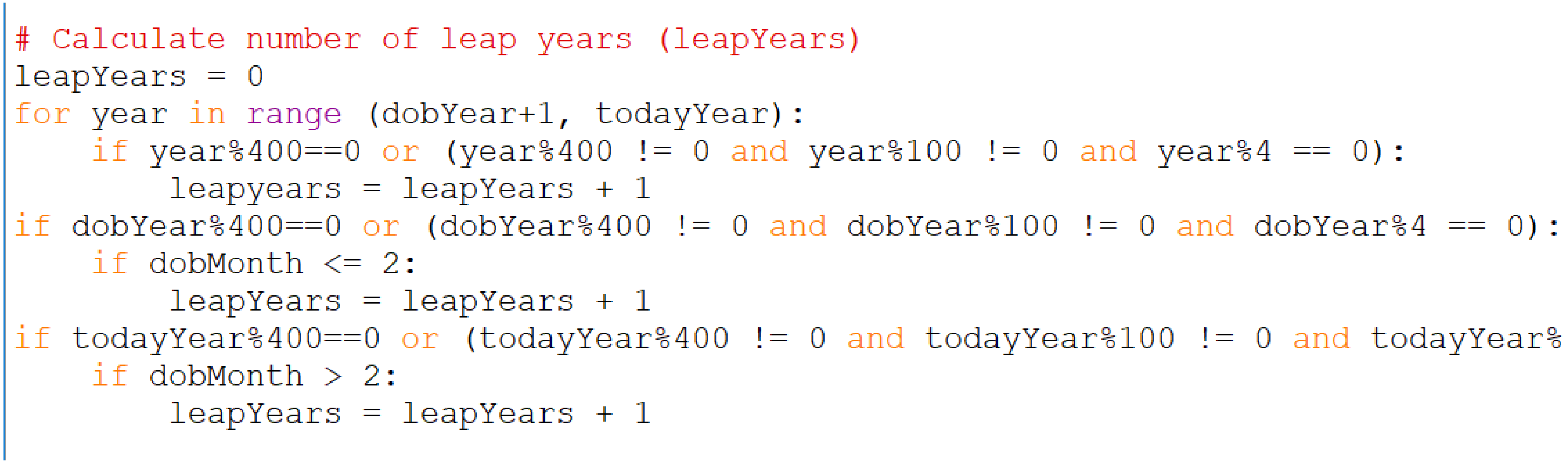
for ... in loops are called determinate or counted loops

```
for <variable> in <iterable>:
        <statement>
        <statement> ...
```

e.g.

```
for number in range(0, 7):
        print(number)
```

Version2



```python
# Calculate number of leap years (leapYears)
leapYears = 0
for year in range (dobYear+1, todayYear):
    if year%400==0 or (year%400 != 0 and year%100 != 0 and year%4 == 0):
        leapyears = leapYears + 1
if dobYear%400==0 or (dobYear%400 != 0 and dobYear%100 != 0 and dobYear%4 == 0):
    if dobMonth <= 2:
        leapYears = leapYears + 1
if todayYear%400==0 or (todayYear%400 != 0 and todayYear%100 != 0 and todayYear%
    if dobMonth > 2:
        leapYears = leapYears + 1
```

leapyears = (todayYear - dobYear)//4

Sequences

The anatomy of programs (so far)

We can think of programs as containing the following:

- **data structures** of different types**,**

  e.g., integers, floating point numbers, Booleans,
  sequences (strings, lists)

- **functions** - actions to be carried out on data

  - e.g., print, input

- **control flow** statements - which redirect the line-by-line flow of code,

  conditionals (if-elif-else)

  iteration (for loops, while loops)

Sequences

Data structures in which elements exist in a strict sequence

strings: a sequence of characters, e.g., 'This is a sentence'

lists: a sequence of data objects

```
fruits = ['apple', 'banana', 'orange', 'grape', 'tomato', 'lemon']

customers = ['Johnson', 'Wang', 'Zhu', 'Agarwal', 'Williams',
             'Nguyen', 'vom Lehn', 'Wilson', 'Trudeau']
```

Similar operations and methods apply to sequences, as we will see.

String operations

Deciding if a password is secure

An organization defines a secure password as follows:
It must be 8 - 20 characters, and it must contain at least three of the character categories among the following:

- Uppercase characters (A-Z)
- Lowercase characters (a-z)
- Digits (0-9)
- Special characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Imagine three people entered the following:

iweYdUhOpLkjhUAU

drdXYe#

y&ui8rto

How would we write a program to take in passwords and report back of they follow the rules or not?

indexing & slicing

Exercise 1

'iweYdUhOpLkjhUAU'

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

```
password = 'iweYdUhOpLkjhUAU'
```

indexing

```
print( password[3] )
print( password[-1] )
```

index

slice

slicing

```
print( password[4:8] )
print( password[:5] )
print( password[-3:] )
print( password[-7:-1] )
print( password[-7:len(password)] )
print( password[:] )
```

Exercise 2

indexing & slicing

'Enter your name: '

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

```
prompt = 'Enter your name:    '
print(    .... ?
```

'Enter'

'your'

'e'

'r n'

'your name'

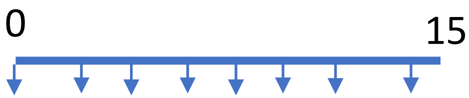
':   '

'ton'

Exercise 3

Slicing: Step Size and Direction

Slicing can specify a third number which indicates how much to step through the sequence by:



```
sentence = "the quick brown fox jumped over the lazy dog"
print( sentence[0:16:2] )
```

slice   step

```
print( sentence[16:0:-2] )
```

String operations

operator	description
+	concatenation
*	repetition
<string>[ ]	indexing
<string>[:]	slicing
len(<string>)	length
for <var> in <string>	iteration through characters

Reference. John Zelle (2017) Python Programming 3rd Ed. Franklin Beedle

Deciding if a password is secure

An organization defines a secure password as follows:
It must be 8 - 20 characters, and it must contain at least three of the character categories among the following:

- Uppercase characters (A-Z)
- Lowercase characters (a-z)
- Digits (0-9)
- Special characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Imagine three people entered the following:

iweYdUhOpLkjhUAU

drdXYe#

y&ui8rto

How would we write a program to take in passwords and report back of they follow the rules or not?

Using a for .. in loop to examine the string

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

```
for i in range(1,6):
        print(i)
```

```
password = 'drdXYe#'
for i in range(len(password)):
        print( i, password[i] )
```

```
password = 'drdXYe#'
for ch in password:
        print(ch )
```

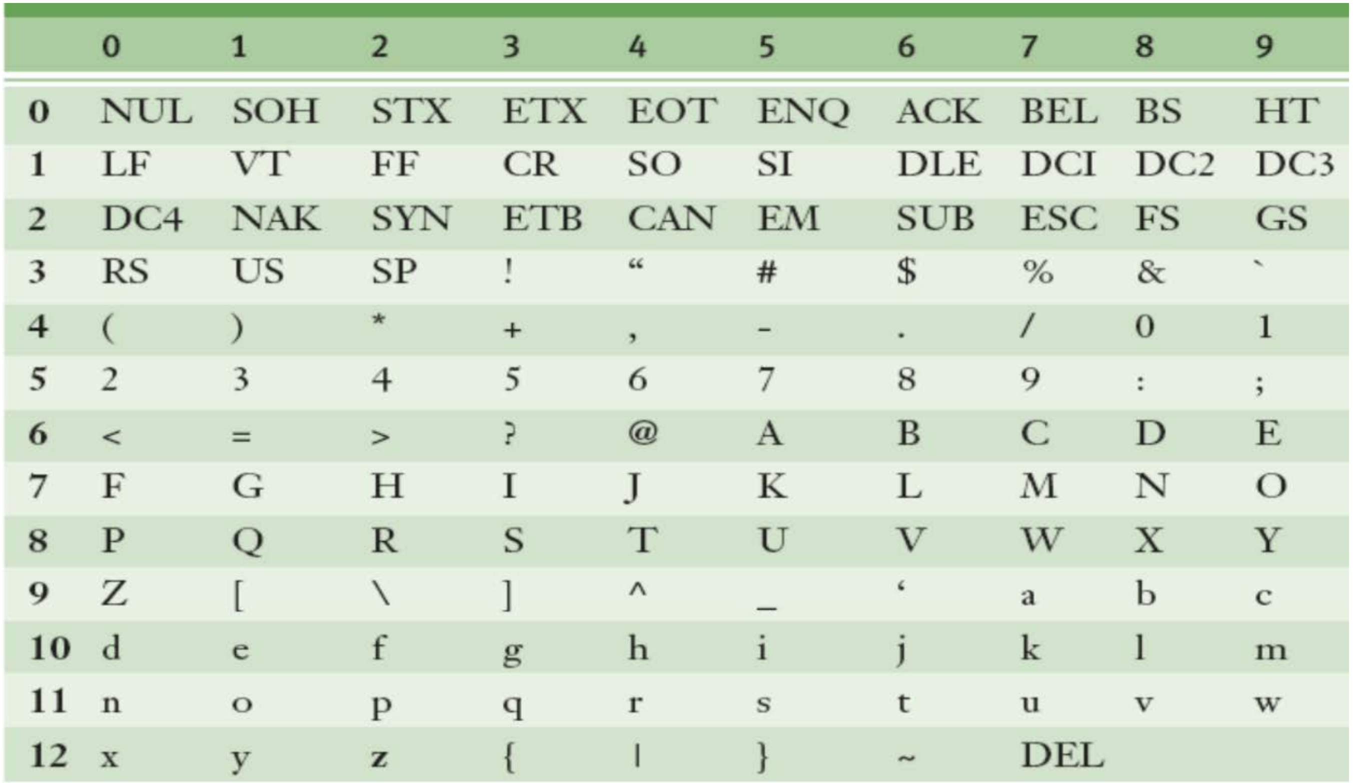
Exercise 4

Exercise 5: Using a for .. in loop

Write a program to report how many lower case letters are in a password.

```
password = 'drdXYe#'
for ch in password:
        print( ch )
```

```
Enter password = 'drdXYe#'
'Number of lower case letters = 4
```



	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	"	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

```
>>> ord(‘a’)
    97
>>> chr(100)
    ‘d’
```

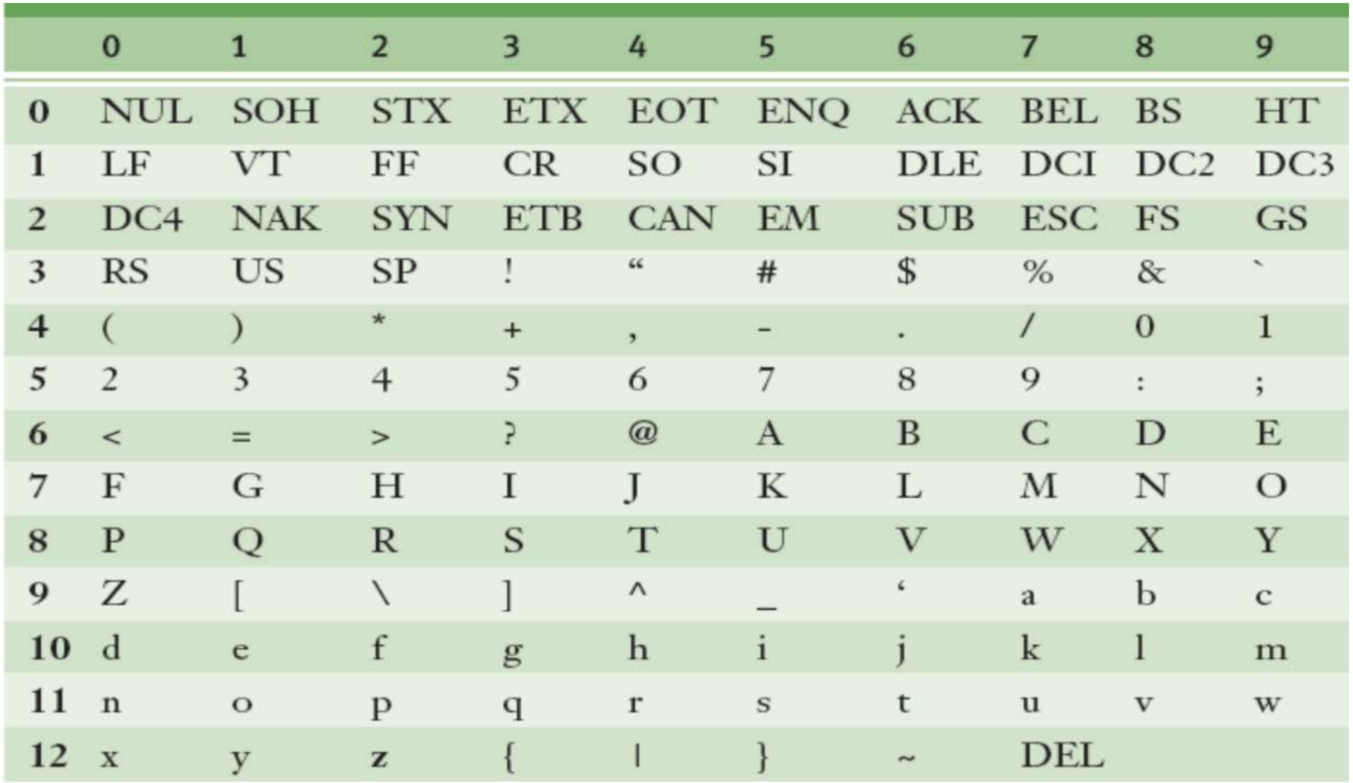
Exercise 6: Using a for .. in loop

Write a program to print out a list of characters in a password with an indication for each character if it is a lower case letter.

```
password = input('Enter password: ')
for ch in password:
        if ord(ch)>=97 and ord(ch)<=122:
```

Enter password = 'drdXYe#'

```
d  lower case
r  lower case
d  lower case
X
Y
e  lower case
#
```



	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

```
>>> ord(‘a’)
 97
>>> chr(100)
 ‘d’
```

Lists

Lists

- Lists are a very powerful data structure
- They can contain a mixture of element types, including other lists!
- Normally populated with data generated or gathered by the program

```
customers = ['Johnson', 'Wang', 'Zhu', 'Agarwal', 'Williams',
                              'Nguyen', 'vom Lehn', 'Wilson', 'Trudeau']

months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
'August', 'September', 'October', 'November', 'December']

profile = [ 'Wilson', 45, True, [2, 4, 56, 23], 'Sydney', 3078]

profile =[name, number, availability, offices, city, zipcode]
```

indexing and slicing (same as for strings)

```
months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
      'August', 'September', 'October', 'November', 'December']
```

```
print( len(months) )
print(months[ 3:5 ] )
```

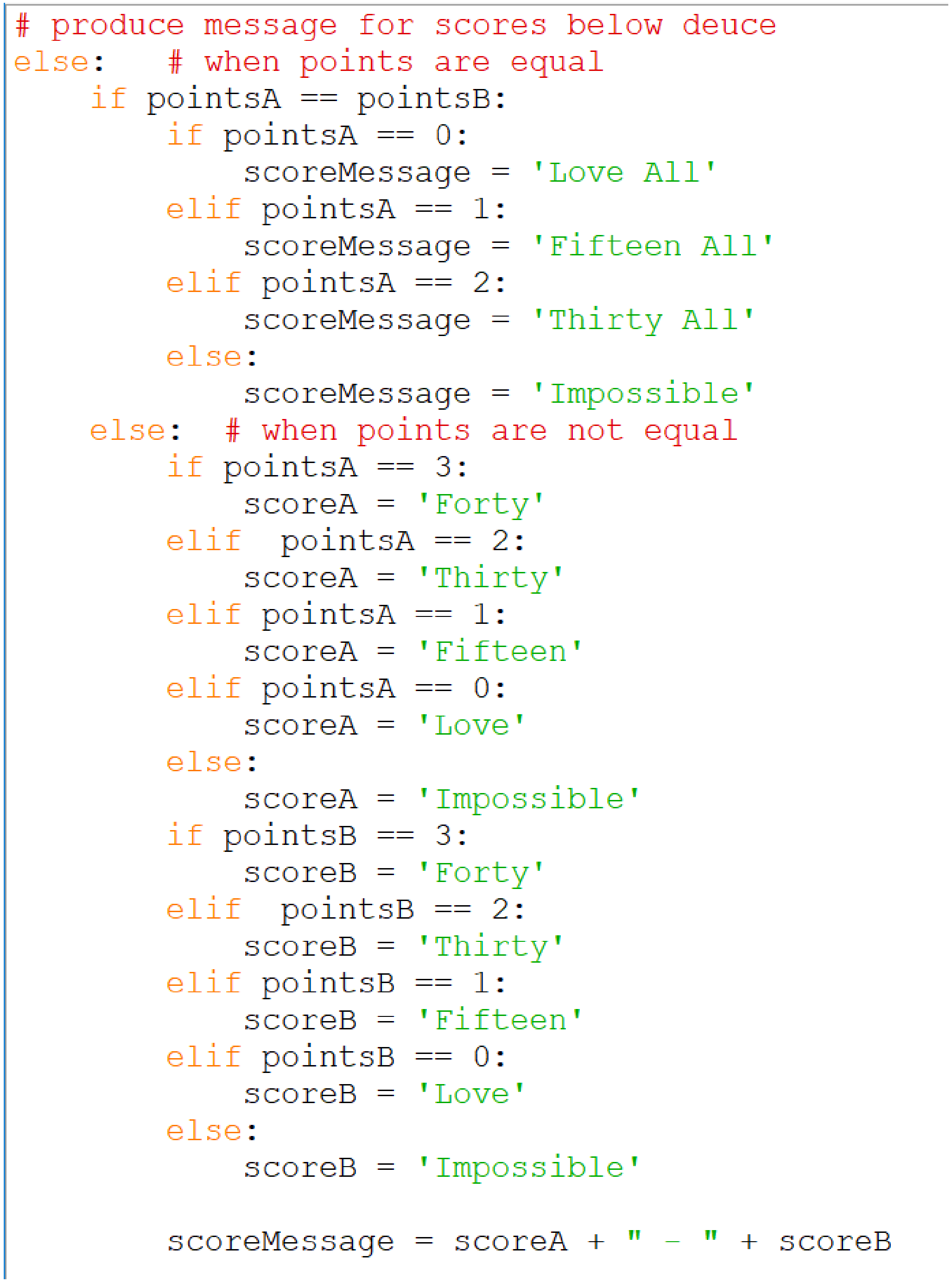
```
print ( months[10])  print(months[-2] )
print ( months[8:11])
print (months[0] )
print( months[2:12:3] )
```

Exercise 7

L3 Ex10: Scoring a game of tennis. Part 2

Write a program that takes in the number of points that two tennis players have scored against each other, and gives the game score so far.

Player A	Player B	Score
0	0	Love All
1	0	Fifteen - Love
1	2	Fifteen - Thirty
2	2	Thirty All
2	3	Thirty - Forty
2	4	Player B wins
3	3	Deuce
3	4	Advantage Player B
4	4	Deuce
4	5	Advantage Player B
6	5	Advantage Player A
7	5	Player A wins
9	11	Player B wins



```
# produce message for scores below deuce
else:   # when points are equal
    if pointsA == pointsB:
        if pointsA == 0:
            scoreMessage = 'Love All'
        elif pointsA == 1:
            scoreMessage = 'Fifteen All'
        elif pointsA == 2:
            scoreMessage = 'Thirty All'
        else:
            scoreMessage = 'Impossible'
    else:  # when points are not equal
        if pointsA == 3:
            scoreA = 'Forty'
        elif  pointsA == 2:
            scoreA = 'Thirty'
        elif pointsA == 1:
            scoreA = 'Fifteen'
        elif pointsA == 0:
            scoreA = 'Love'
        else:
            scoreA = 'Impossible'
        if pointsB == 3:
            scoreB = 'Forty'
        elif  pointsB == 2:
            scoreB = 'Thirty'
        elif pointsB == 1:
            scoreB = 'Fifteen'
        elif pointsB == 0:
            scoreB = 'Love'
        else:
            scoreB = 'Impossible'

        scoreMessage = scoreA + " - " + scoreB
```

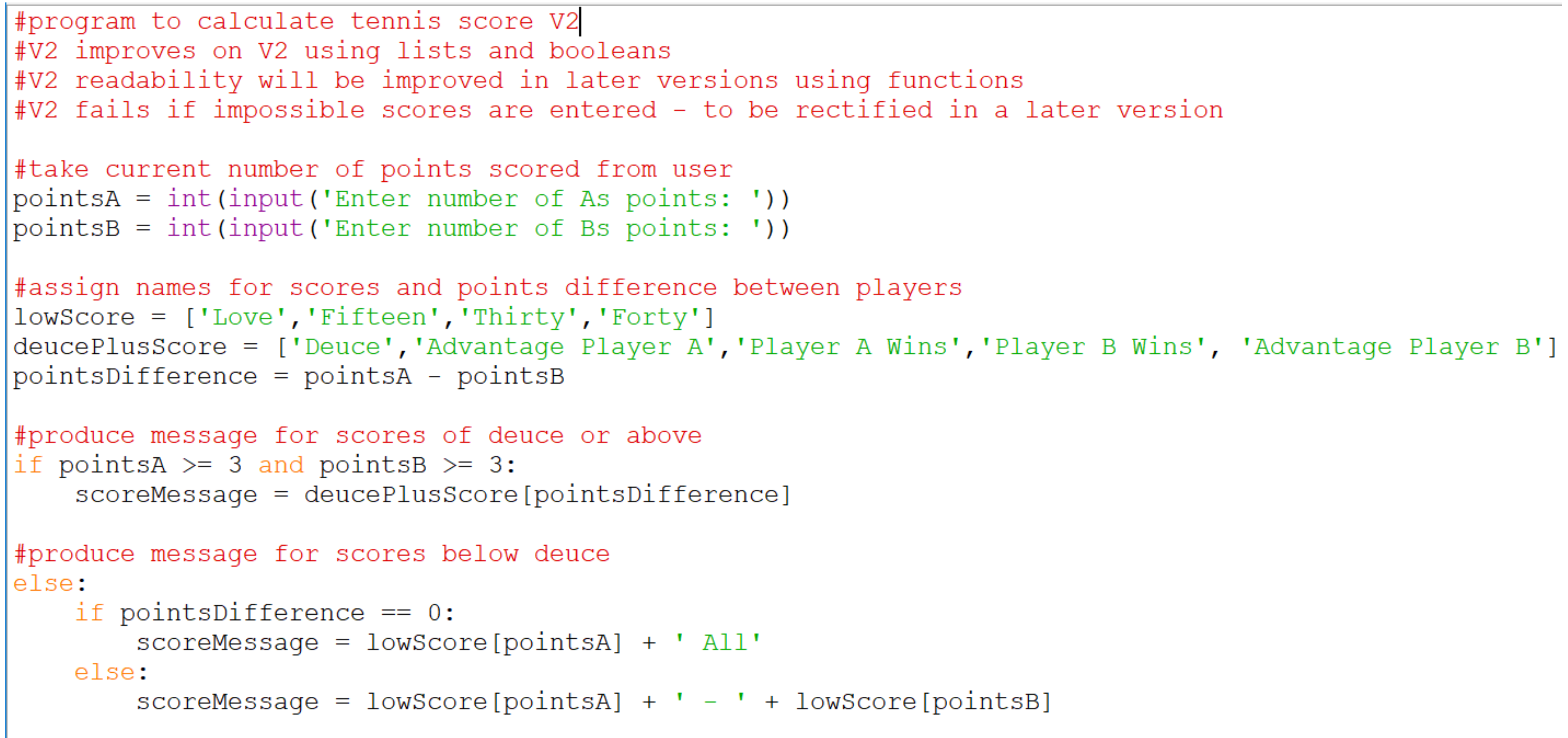
#V1 produces a correct score

# ... but V1 is not elegant

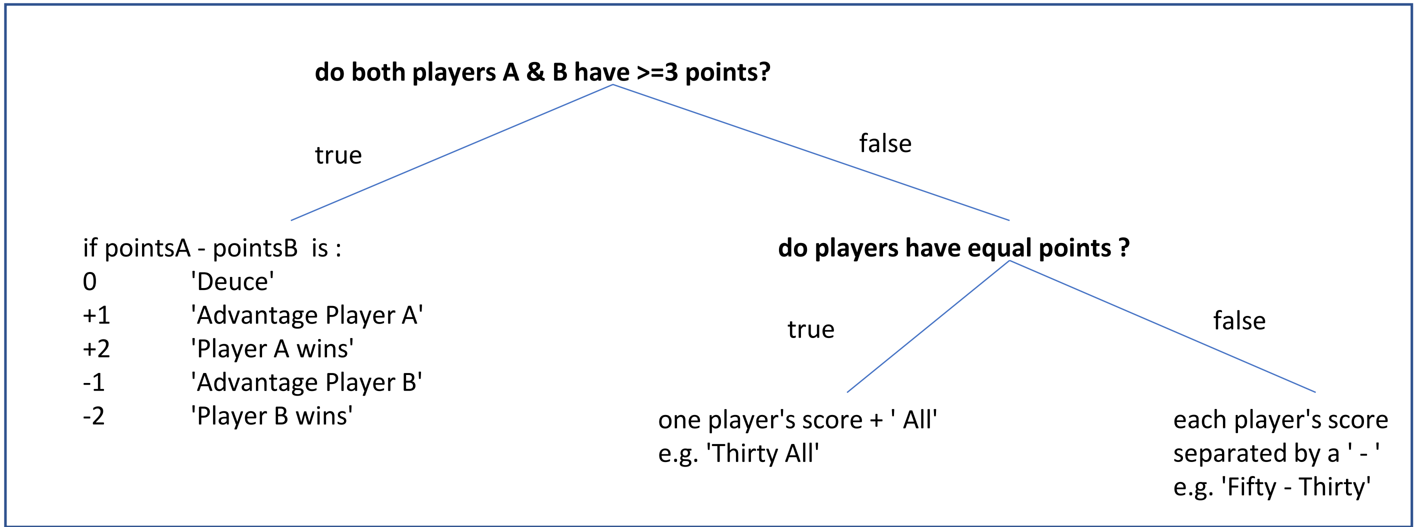
# ... in fact is terrible! ... it is long, convoluted and hard to comprehend

# ... commenting cannot help this disaster because the structure of the program is confusing

#Later versions will make a readable program using booleans, lists and functions



```
#program to calculate tennis score V2
#V2 improves on V2 using lists and booleans
#V2 readability will be improved in later versions using functions
#V2 fails if impossible scores are entered - to be rectified in a later version

#take current number of points scored from user
pointsA = int(input('Enter number of As points: '))
pointsB = int(input('Enter number of Bs points: '))

#assign names for scores and points difference between players
lowScore = ['Love','Fifteen','Thirty','Forty']
deucePlusScore = ['Deuce','Advantage Player A','Player A Wins','Player B Wins', 'Advantage Player B']
pointsDifference = pointsA - pointsB

#produce message for scores of deuce or above
if pointsA >= 3 and pointsB >= 3:
    scoreMessage = deucePlusScore[pointsDifference]

#produce message for scores below deuce
else:
    if pointsDifference == 0:
        scoreMessage = lowScore[pointsA] + ' All'
    else:
        scoreMessage = lowScore[pointsA] + ' - ' + lowScore[pointsB]

#report score message
print(scoreMessage)
```



do both players A & B have >=3 points?

true

if pointsA - pointsB is :

0	'Deuce'
+1	'Advantage Player A'
+2	'Player A wins'
-1	'Advantage Player B'
-2	'Player B wins'

false

do players have equal points ?

true

one player's score + ' All'
e.g. 'Thirty All'

false

each player's score
separated by a ' - '
e.g. 'Fifty - Thirty'

String methods

Objects & Methods

In addition to:

data structure, **functions**, **control flow** statements

... programs also contain:

objects - in Python, all data values are objects

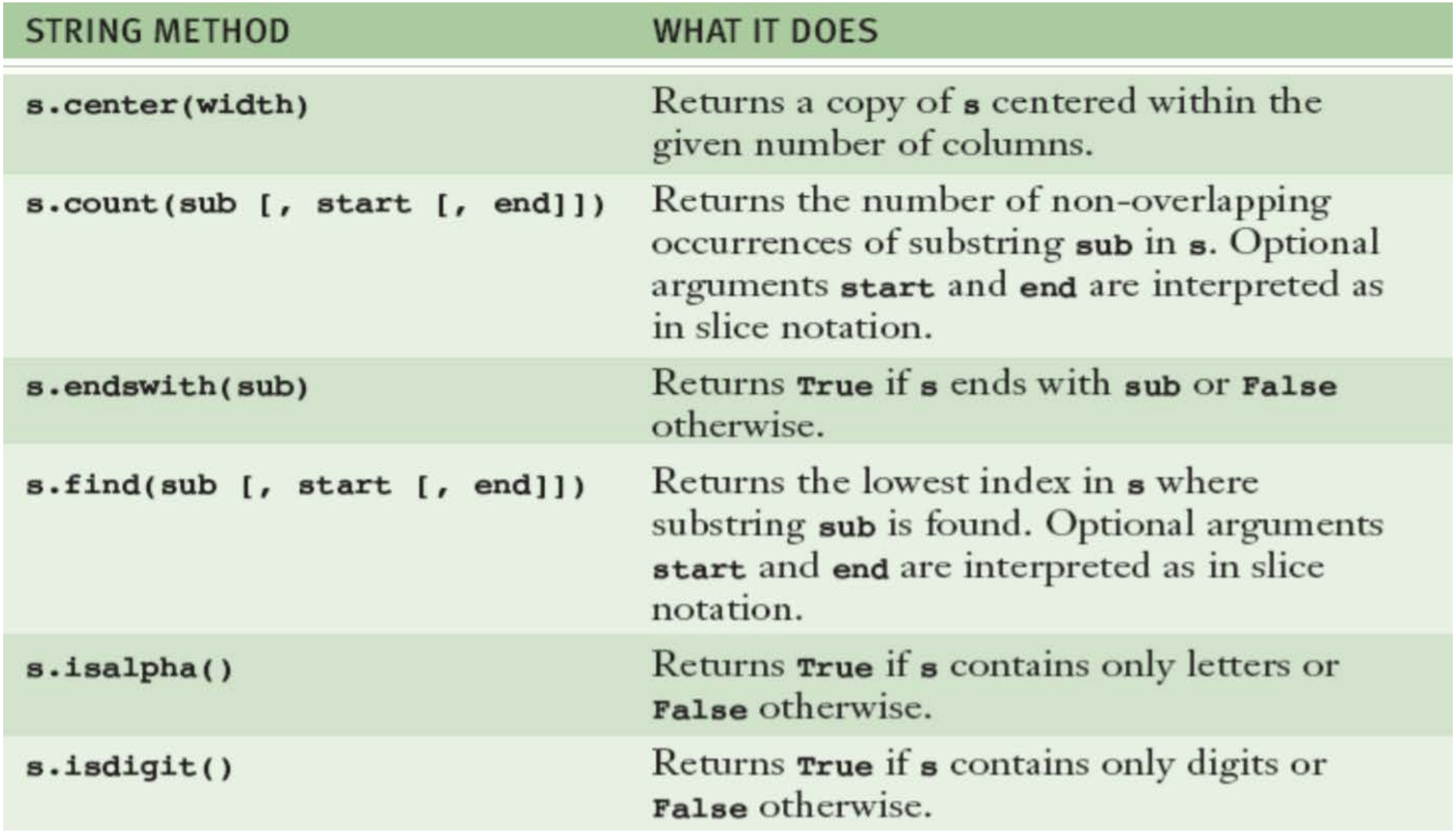
methods - a kind of function that belong to **objects** of a particular type, and can be only be performed on those objects.

e.g., **upper()** is a method that belongs to string objects ...

```
name = 'Jack Darling'
print(name.upper() )
```

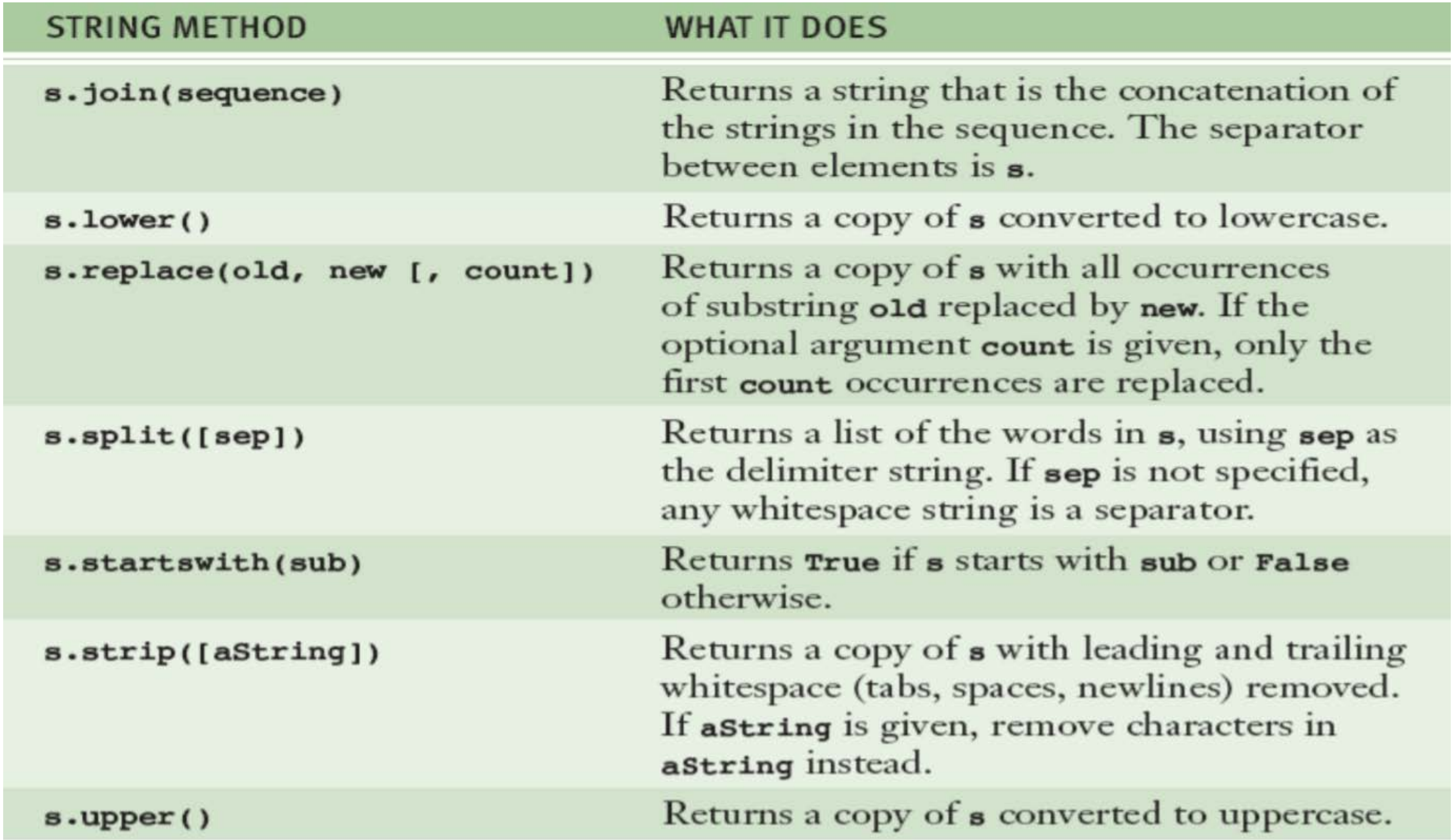
dir(str) or help(str) will give you the entire list of string methods

String Methods (continued)



STRING METHOD	WHAT IT DOES
`s.center(width)`	Returns a copy of `s` centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring `sub` in `s`. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.endswith(sub)`	Returns `True` if `s` ends with `sub` or `False` otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in `s` where substring `sub` is found. Optional arguments `start` and `end` are interpreted as in slice notation.
`s.isalpha()`	Returns `True` if `s` contains only letters or `False` otherwise.
`s.isdigit()`	Returns `True` if `s` contains only digits or `False` otherwise.

String methods (continued)



STRING METHOD	WHAT IT DOES
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is `s`.
`s.lower()`	Returns a copy of `s` converted to lowercase.
`s.replace(old, new [, count])`	Returns a copy of `s` with all occurrences of substring `old` replaced by `new`. If the optional argument `count` is given, only the first `count` occurrences are replaced.
`s.split([sep])`	Returns a list of the words in `s`, using `sep` as the delimiter string. If `sep` is not specified, any whitespace string is a separator.
`s.startswith(sub)`	Returns `True` if `s` starts with `sub` or `False` otherwise.
`s.strip([aString])`	Returns a copy of `s` with leading and trailing whitespace (tabs, spaces, newlines) removed. If `aString` is given, remove characters in `aString` instead.
`s.upper()`	Returns a copy of `s` converted to uppercase.

Exercise 8

String method Examples

```
imagine = "You may say I'm a dreamer”
print( imagine.split() )

print( '-'.join(imagine.split()) )
msg = "I love apples, apple are my favorite fruit"

print( msg.count("apple") )

x = 'foobar’
print( x.endswith('bar’) )

print( x.startswith(‘bar’) )

print( imagine.isalpha() )
```

String method parameters

Exercise 9

methods have parentheses which may contain parameters
e.g. **split([sep])**

This method splits a string into substrings, breaking it at the specified separator character [sep], or at the default space character.

```
print('what is the point of these methods?'.split()  )
```

```
print('what is the point of these methods?'.split('e')  )
```

```
name = input('Enter you full name: ').split()
print(name)
>>> Enter your full name: David Robert Jones
```

Exercise 10 : Using split in the daysOfLife program

```
# Get birthdate from user as 3 values: year, month & day
dobYear = int(input('Enter year of birth :'))
dobMonth = int(input('Enter month of birth (1 -12) :'))
dobDay = int(input('Enter day of birth: '))
```

Re-write the opening part of the days of life code (above) using the split method, so that there is just ONE input statement and it asks the user to enter their birthdate in the format: DD/MM/YYYY e.g. 06/03/2006

(Tip: you can also use simultaneous assignment)

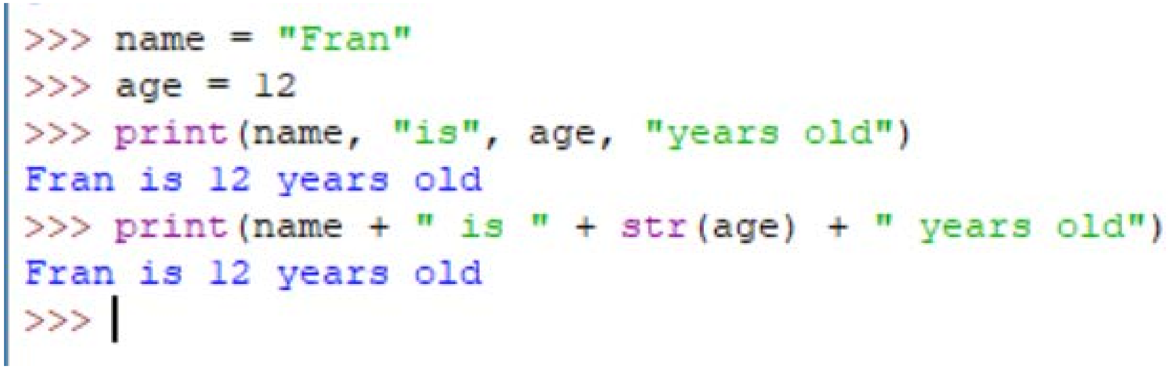
List methods

method	description
<list>.append(x)	add element x to end of list
<list>.sort()	sorts the list into order
<list>.reverse()	reverses elements in the list
<list>.index(x)	returns the index of first occurrence of x in the list
<list>.insert(i, x)	inserts x into the list at index i
<list>.count(x)	returns the number of occurrences of x in the list
<list>.remove(x)	deletes the first occurrence of x in the list
<list>.pop(i)	deletes the i-th element in the list and returns its value

Reference. John Zelle (2017) Python Programming 3rd Ed. Franklin Beedle

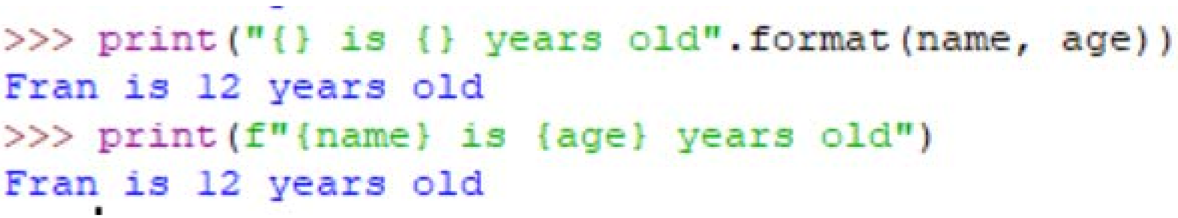
String formatting

- To date when joining strings and literals we have used the string concatenation symbol “+” or “,”



```
>>> name = "Fran"
>>> age = 12
>>> print(name, "is", age, "years old")
Fran is 12 years old
>>> print(name + " is " + str(age) + " years old")
Fran is 12 years old
>>> 
```

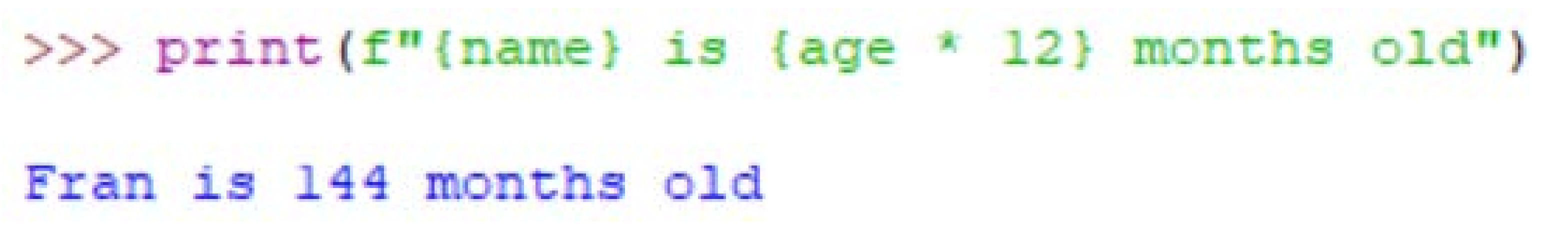
- str.format() and the newer f-strings (formatted string literals) allow for greater control and functionality when formatting strings



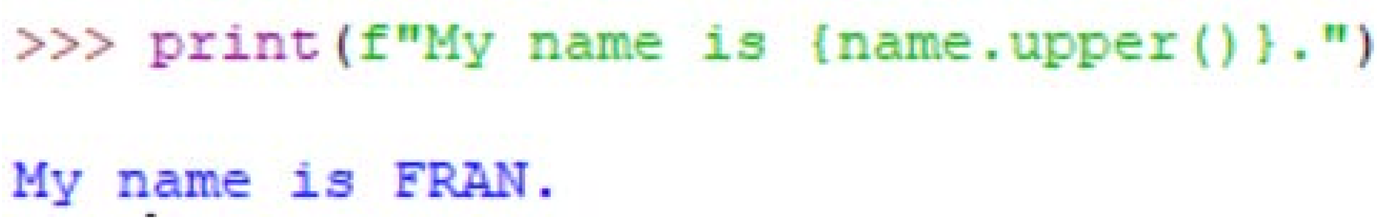
```
>>> print("{} is {} years old".format(name, age))
Fran is 12 years old
>>> print(f"{name} is {age} years old")
Fran is 12 years old
```

String formatting continued

- Inside a “string literal” curly brackets are used as placeholders for expressions that will be replaced by their values
- f-strings are evaluated at runtime and all valid Python expressions are allowed
- You call also call Python methods



```
>>> print(f"{name} is {age * 12} months old")

Fran is 144 months old
```



```
>>> print(f"My name is {name.upper()}.")

My name is FRAN.
```

Challenge 1: Making the DaysOfLife program elegant

So far we have made the Day Of Life program accurate using conditionals (if ... elif ... else) in Version 2

But Version 2 is NOT elegant - it is hard to comprehend the complex if ... structure.

Simplify the DaysOfLife program code and make it more readable using techniques we have learnt in this lecture, chiefly the use of Lists.

Tip: Look at the parts of the code in V2 where there is a long list of ifs. Consider how we could replace the literal values with lists and indexing to get the right element in the list. You will need to define the lists and their values at the beginning of your program. See how this was done for the tennis scoring program V2 for some ideas.

Challenge 2: The Password Security problem

An organization defines a secure password as follows:
It must be 8 - 20 characters, and it must contain at least three of the character categories among the following:

- Uppercase characters (A-Z)
- Lowercase characters (a-z)
- Digits (0-9)
- Special characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Write a program to take in passwords and report back if they follow the rules or not?

TIP: Look at the exercises we did in class that related to this problem. In one of those exercise we used a variable to count the number of letters in a password. Think how you can use a similar technique here to count how many categories are included in the password.