

Lecture 5: Functions

COMP90059 Introduction to Programming

Wally Smith

School of Computing & Information Systems

Lecture Overview

- Reviewing techniques and challenges from Lecture 4 - you will need to be sure about these to tackle the material in this lecture
 - Warm Up Exercises on basic Sequence Indexing and Slicing
 - Warm Up Exercises on For loops
 - Reviewing Booleans from Lecture 2
 - Using lists to improve the elegance of the Days of Life program
 - Developing the Password Security program
- Functions
 - The power of problem decomposition
 - Function definition syntax
 - Parameter and arguments
 - Scope: local and global
 - The art of modular program design

Reviewing and Extending Techniques from Lecture 4 on Sequences

Lecture 4: indexing & slicing

'iweYdUhOpLkjhUAU'

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

```
password = 'iweYdUhOpLkjhUAU'
```

indexing

index

slice

slicing

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

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

```
print( password[4:8] )
```

```
print( password[:5] )
```

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

```
print( password[-7:-1] )
```

```
print( password[-7:len(password)] )
```

```
print( password[:] )
```

Y

U

dUhO

iweYd

UAU

LkjhUA

LkjhUAU

weYdUhOpLkjhUAU

Warm-Up Exercise 1: indexing & slicing sequences

```
customers = 'JeffSarahColinArmind'
```

```
print(customers[3:7])
```

```
print(customers[-9:])
```

```
print(customers[::2])
```

```
print(len(customers[2:6]))
```

```
m = 2
```

```
print(customers[:m+3])
```

```
customers = ['Jeff', 'Sarah', 'Colin', 'Armind']
```

```
print(customers[1:3])
```

```
print(customers[2:])
```

```
print(customers[-1])
```

Lecture 4: for ... in loops

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(7):  
    print(number)
```

0
1
2
3
4
5
6

Lecture 4: Using a for .. in loop

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

0
1
2
3
4

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

1
2
3
4
5

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

0 d
1 r
2 d
3 X
4 Y
5 e
6 #

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

d
r
d
X
Y
e
#

Warm-Up Exercise 2: for ... loops

Write a program using a for ... loop to generate a list of numbers, from 1 to 1000 inclusive

Write a program to take in a word from the user, and then print out the letters of the word in a list.

Warm-Up Exercise 3: nested for ... loops

A. Complete the program below so that it prints out the names of each month of the year in a continuous list down the page.

B. Modify your program below so that it prints out the letters of all the month names in one continuous stream.

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

```
January  
February  
March  
April  
May  
June  
July  
August  
September  
October  
November  
December
```

J
a
n
u
a
r
y
F
e
b
r
u
a
r
y
M
a
r
c
h
A
p
r
i
l
M
a
y
J
u
n
e
J
u
l
y

Lecture 2: Data Types

| Type of data | Python type name | Examples (literals) |
|-------------------|------------------|-------------------------------|
| Integers | int | -1, 0, 6895 |
| Real numbers | float | -0.101011, 567738.009187 |
| Character strings | str | ""', 'd', "I'm a string", '3' |
| Boolean | bool | True, False |

- 'The data type of an item determines what values it can hold and what operations it supports' (Zelle, 2017)
- A literal is a specific or 'actual' data value

Boolean (bool)

- Booleans are variables that either true or false
Examples: `PaidParkingTicket = True`, `GoalAchieved = False`
- Python represents booleans in data type: `bool`
- Booleans literals are: `True` and `False` (note the capitalization)
- Boolean variables or Boolean expressions are the conditions that determine decisions:
e.g., `if ParkingTicketPaid:`
 `print('Thank you!')`

Revisiting Challenges from Lecture 4

Analyzing the days of life calculation

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



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

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.

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')

Version 3

```
# Define days of each month
```

```
daysOfMonths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

Version 2

```
# Calculate days in incomplete current year of life (daysCurrentYear)
```

```
daysCurrentYear = todayDay
```

```
for monthDays in daysOfMonths[:todayMonth-1]:
```

```
    daysCurrentYear += monthDays
```

```
if todayMonth == 1:
    daysBeforeMonth = 0
elif todayMonth==2:
    daysBeforeMonth = 31
elif todayMonth==3:
    daysBeforeMonth = 31+
elif todayMonth==4:
    daysBeforeMonth = 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
```

Imagine if current date is 16th April 2020 ...

- first set daysCurrentYear = 16 days
- for each of the preceding months 0, 1, 2 ...
- ... add its number of days to daysCurrentYear

= 16

+ 31 + 28 + 31

= 106

Version 3

```
# Define days of each month
daysOfMonths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

# Calculate days in incomplete first year of life (daysFirstYear)
daysFirstYear = -dobDay
for monthDays in daysOfMonths[dobMonth-1:]:
    daysFirstYear += monthDays
```

Version 2

```
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
elif dobMonth==7:
    daysFirstMonth, daysAfterMonth = 31, 365-31-28-31-30-31
elif dobMonth==8:
    daysFirstMonth, daysAfterMonth = 31, 365-31-28-31-30-31
elif dobMonth==9:
    daysFirstMonth, daysAfterMonth = 30, 365-31-28-31-30-31
elif dobMonth==10:
    daysFirstMonth, daysAfterMonth = 31, 365-31-28-31-30-31
elif dobMonth==11:
    daysFirstMonth, daysAfterMonth = 30, 365-31-28-31-30-31-30-31-30-31-30
else: #todayMonth==12
    daysFirstMonth, daysAfterMonth = 31, 0
```

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

Imagine if current dob is 10 October 2001

- first set daysFirstYear to -10 days
- for each of the months from October onwards: 9, 10, 11 ...
- ... add its number of days to daysFirst Year
 - - 10
 - + 31 + 30 + 31
 - = 82 days

Version 3

```
# Define days of each month
daysOfMonths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

# Calculate days in incomplete first year of life (daysFirstYear)
daysFirstYear = -dobDay
for monthDays in daysOfMonths[dobMonth-1:]:
    daysFirstYear += monthDays

# Calculate how many days in whole years of life (daysWholeYears)
daysWholeYears = (todayYear - dobYear - 1) * 365

# Calculate days in incomplete current year of life (daysCurrentYear)
daysCurrentYear = todayDay
for monthDays in daysOfMonths[:todayMonth-1]:
    daysCurrentYear += monthDays

# 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:
        if dobYear!=todayYear: #don't add another leapYear if born in current ye
            leapYears = leapYears + 1

# Calculate days of Life by summing its four components
daysOfLife = daysFirstYear + daysWholeYears + daysCurrentYear + leapYears

# Report days of life to user
print('You have been alive for', daysOfLife, 'days')
```

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.

The Password Security problem: Analysis and high-level design

#get password from user (let's imagine it is UHHiii45klm64)

count the number of characters in each category (specials, digits, lower case, upper case)

#count the number of categories that have greater than 0

#set status of password: length>8 and length<20 and categories>2

#report status to user

Version 0

```
#get password from user
password = input('Enter your proposed password: ')

#count the number of characters in each category
special = 0
digital = 0
lowerCase = 0
upperCase = 0
for character in password:
    o = ord(character)
    if o>=33 and o<=47:
        special = special + 1
    if o>=48 and o<=57:
        digital = digital + 1
    if o>=97 and o<=122:
        lowerCase = lowerCase + 1
    if o>=65 and o<=90:
        upperCase = upperCase + 1

#count the number of categories that have greater than 0
numberCategories = 0
if special>0:
    numberCategories = numberCategories + 1
if digital>0:
    numberCategories = numberCategories + 1
if lowerCase>0:
    numberCategories = numberCategories + 1
if upperCase>0:
    numberCategories = numberCategories + 1

#set status of password
if len(password)>= 8 and len(password)<=20 and numberCategories>2:
    status = 'secure'
else:
    status = 'not secure'

#report status to user
print(password, 'is', status)
```

Version 1

```
#get password from user
password = input('Enter your proposed password: ')

#test for presence of specials, digits, lowerCase, upperCase
special = False
digital = False
lowerCase = False
upperCase = False
for character in password:
    o = ord(character)
    if o>=33 and o<=47:
        special = True
    if o>=48 and o<=57:
        digital = True
    if o>=97 and o<=122:
        lowerCase = True
    if o>=65 and o<=90:
        upperCase = True

#count the number of character categories detected
numberCategories = 0
if special:
    numberCategories += 1
if digital:
    numberCategories += 1
if lowerCase:
    numberCategories += 1
if upperCase:
    numberCategories += 1

#set status of password
if len(password)>= 8 and len(password)<=20 and numberCategories>2:
    status = 'secure'
else:
    status = 'not secure'

#report status to user
print(password, 'is', status)
```

#program to tell if a password is secure Version 2

```
#get password from user
password = input('Enter your proposed password: ')

#set up characters types
special = [False,33,47]
digit = [False,49,57]
lowerCase = [False,97,122]
upperCase = [False,65,90]

chartypes = [special, digit, lowerCase, upperCase]

#test for presence of specials, digits, lowerCase, upperCase
for char in password:
    for chartype in chartypes:
        if ord(char) in range(chartype[1],chartype[2]+1):
            chartype[0]= True

#count the number of categories that are present
numberCategories = 0
for chartype in chartypes:
    if chartype[0]:
        numberCategories +=1

#check the length of password
if len(password)>= 8 and len(password)<=20:
    lengthOK = True
else:
    lengthOK = False

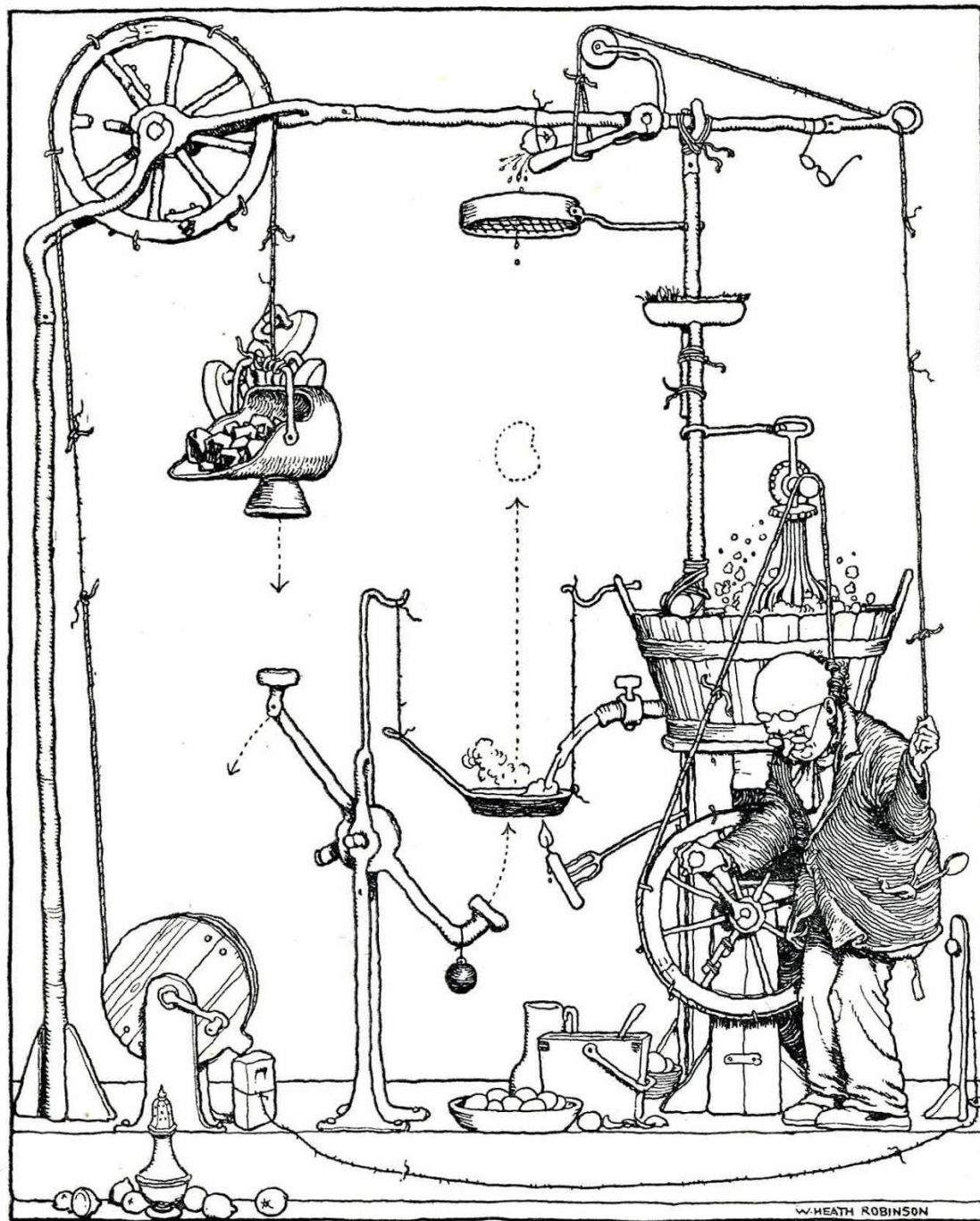
#set status of password
if lengthOK and numberCategories>2:
    status = 'secure'
else:
    status = 'not secure'

#report status to user
print(password,'is',status)
```

```
#test for presence of specials,
special = False
digital = False
lowerCase = False
upperCase = False
for character in password:
    o = ord(character)
    if o>=33 and o<=47:
        special = True
    if o>=48 and o<=57:
        digital = True
    if o>=97 and o<=122:
        lowerCase = True
    if o>=65 and o<=90:
        upperCase = True
```

```
#count the number of character
numberCategories = 0
if special:
    numberCategories += 1
if digital:
    numberCategories += 1
if lowerCase:
    numberCategories += 1
if upperCase:
    numberCategories += 1
```

Functions



Problem decomposition

Problem decomposition is the way we take a complex problem or task (that is difficult to solve) and break it down into smaller components (each one being easier to solve than the whole).

The secret of good programs is effective problem decomposition, also known as **stepwise refinement**.

Programs decompose the task through **modularity** - meaning the program is made up of separate **functions** of code - each solving a part of the overall task.

An effective program design creates functions that are **loosely coupled** - meaning that they have clearly defined inputs and outputs which are kept as simple as possible.

Functions - how do they help program design?

Defining functions is a core technique of programming.

Functions are like sub-programs within the main program.

Organising programs as a set of functions supports problem decomposition.

Functions thereby allow us to create programs with:

- **elegance** (by breaking down complexity into simple parts)
- **readability** (by meaningful function names)
- **traceability** (by reducing code duplication)
- **extensibility** (by providing building blocks of code)

Lecture 1 # Take the user's name and produces the text of the Happy Birthday song for that person.

```
name = input('Enter your name: ')  
print('Happy Birthday to you')  
print('Happy Birthday to you')  
print('Happy Birthday, dear', name, '!')  
print('Happy Birthday to you')
```

```
>>>>Enter your name: Donald
```

```
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday, dear Donald !  
Happy Birthday to you
```

```
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday, dear Donald !  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday, dear Joshua !  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday, dear Cindy !  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday, dear Haoyang !  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday to you  
Happy Birthday, dear Indira !  
Happy Birthday to you
```

program to produce happy birthday song

define a function to print out the song for a named person

```
def singFor(name):  
    print('Happy Birthday to you')  
    print('Happy Birthday to you')  
    print('Happy Birthday, dear', name, '!')  
    print('Happy Birthday to you')
```

call the function for many people

```
singFor('Donald')  
singFor('Joshua')  
singFor('Cindy')  
singFor('Haoyang')  
singFor('Indira')
```

the following will do the exactly the same


```
people = ['Donald', 'Joshua', 'Cindy', 'Haoyang', 'Indira']  
for person in people:  
    singFor(person)
```

Happy Birthday to you
Happy Birthday to you
Happy Birthday, dear Donald !
Happy Birthday to you
Happy Birthday to you
Happy Birthday to you
Happy Birthday, dear Joshua !
Happy Birthday to you
Happy Birthday to you
Happy Birthday to you
Happy Birthday, dear Cindy !
Happy Birthday to you
Happy Birthday to you
Happy Birthday to you
Happy Birthday, dear Haoyang !
Happy Birthday to you
Happy Birthday to you
Happy Birthday to you
Happy Birthday, dear Indira !
Happy Birthday to you

Function definition - syntax


```
def functionName(parameter1, parameter2 ...):  
    statement 1  
    statement 1  
    ...
```

'defining'
the
function



```
def greet(name):  
    print('Hello', name)  
    print('How are you going?')
```

'calling'
the
function
from
anywhere
in the
program




```
greet('Joshua')  
greet('Cindy')  
greet('Haoyang')  
greet('Indira')
```

```
Hello Joshua  
How are you going?  
Hello Cindy  
How are you going?  
Hello Haoyang  
How are you going?  
Hello Indira  
How are you going?
```

Parameters and arguments

```
def greet(name, age, address):  
    print('Hello', name)  
    print('Please confirm that we have your details correct:')  
    print('Age:', age, 'years')  
    print('Address:\n', address)
```

parameters
= variables in definition




...

```
greet('Susan', 26, ['2/32 Johnston Street', 'Collingwood', 'VIC 3066'])
```

```
n = 'Susan'  
a = 26  
addr = ['2/32 Johnston Street', 'Collingwood', 'VIC 3066']  
greet(n, a, addr)
```

arguments = values sent when function called



Hello Susan

Please confirm that we have your details correct:

Age: 26 years

Address:

['2/32 Johnston Street', 'Collingwood', 'VIC 3066']

Exercise:

- A. Write a program to print out the names of the drinks: Tea, Coffee, Beer, Wine, Lemonade. Use a function (call it 'announce') that executes the print statement for each drink. Use a list of drinks and remember to put each name in inverted commas.
- B. Modify the function of your program so that it prints out the name of each drink followed by '***' and prints a blank line after each drink.
- C. Modify the function of your program so that it ONLY prints out drinks that have 'e' as the second letter.

Positional Arguments vs Keyword Arguments

So far, we have been providing positional arguments in our function calls. This means that the arguments are in the same order as the original function parameters.

```
def greet(name, age, address)
...
greet('Susan', 26, ['2/32 Johnston Street', 'Collingwood'])
```

Another technique allowed by Python allows is through keywords, allowing a different order.

```
def greet(name, age, address)
...
greet(age =26, address =['2/32 Johnston Street', 'Collingwood'], name ='Susan')
```

Functions - variable scope: local and global

Variables that are created inside a function are called local.

Local variables do not exist outside of the function.

Local may be created with the same name as an outside variable, but they are actually different.

```
def showCube(x):  
    x = x ** 3  
    print('Cube is', x)
```

```
x = 4
```

```
showCube(x)
```

```
print(x)
```

Cube is 64

4

Functions - variable scope: local and global

Global variables are those that are defined outside of any function.

Global variables can be seen by Python inside any function.

Global variables inside a function may be necessary but they reduce readability.

```
def showCube(x):  
    x = x ** 3  
    print('Cube is', x, y)
```

```
y = 'cms cubed'  
x = 4  
showCube(x)  
print(x)
```

Cube is 64 cms cubed
4

Functions - variable scope: local and global

Here is what happens if we try to use a local variable OUTSIDE of the function where it was defined.

```
def showCube(x):  
    y = x ** 3  
    print('Cube is ', y, units)
```

```
units = 'cms cubed'  
x = 4  
showCube(x)  
print(y)
```

Cube is 64 cms cubed

Traceback (most recent call last): File
"test.py", line 8, in <module>
 print(y)
NameError: name 'y' is not defined

functions that return values

If a function calculates a value or a set of values the return technique is very effective to make the code readable.

return - is the reserved word that appears in the function to declare what is returned as the function's result.

```
def cube(x):  
    return x**3  
  
for n in range(10):  
    print('The cube of',n,'is',cube(n))
```

The cube of 0 is 0
The cube of 1 is 1
The cube of 2 is 8
The cube of 3 is 27
The cube of 4 is 64
The cube of 5 is 125
The cube of 6 is 216
The cube of 7 is 343
The cube of 8 is 512
The cube of 9 is 729

functions can return at many points

```
# program to classify people by age

def demographic(name, age):
    if age > 64:
        return 'senior'
    if age > 17:
        return 'adult'
    if age > 12:
        return 'teenager'
    if age > 2:
        return 'young child'
    if age > 0:
        return 'toddler'
    else:
        return 'baby'

people = ['Donald', 'Joshua', 'Cindy', 'Haoyang', 'Indira']
ages = [0, 76, 2, 15, 26]

for person in range(len(people)):
    label = demographic(people[person], ages[person])
    print (people[person], 'is a', label)
```

A warning about lists in functions ...

```
def showCube(x):  
    print(x)  
    x = x ** 3  
    print(x)
```

```
x = 4  
showCube(x)  
print(x)
```

4
64
4

```
def announce(number):  
    print(drinks[number])  
    drinks[number] = 'Whisky'  
    print(drinks[number])
```

```
drinks = ['Tea', 'Coffee', 'Beer', 'Wine', 'Lemonade']  
announce(0)  
print(drinks[0])
```

Tea
Whisky
Whisky

The art of modular program design

The use of functions should reflect the way we have decomposed the overall problem.

Always start by first analyzing the problem and identifying the stages that your program will pass through.

Where a calculation or procedure is used many times, define a function that can be used again and again to execute this step. Where functions calculate a specific value or set of values use the return technique.

In this way, identify separable sub-tasks that can be performed somewhat independently and decide out what inputs it needs (becoming its parameters) and what outputs it will deliver back to the rest of the program.

Give functions (and their parameters) meaningful names that help people to easily understand exactly what it does. A good function name can replace a comment line and make the program 'self-readable'.

What does this code do?

```
principal = eval(input('Enter the initial principal: '))  
apr = eval(input('Enter the annual interest rate: '))  
numberYears = int(input('Enter the number of years: '))  
apr = apr/100  
for i in range(numberYears):  
    principal = principal * (1 + apr)  
print('The final value is: ', round(principal))
```

Note how it is made up of STATEMENTS which are executed by the computer in order.

```
# a program to calculate the future value of a sum
# based on: the initial principal, the annual percentage interest rate & number

def main():
    principal, apr, numberYears = getUserData()
    print('The final value is: ',futureValue(principal, apr, numberYears))

def getUserData():
    principal = eval(input('Enter the initial principal: '))
    apr = eval(input('Enter the annual interest rate: '))
    numberYears = int(input('Enter the number of years: '))
    return principal, apr, numberYears

def futureValue(principal, apr, numberYears):
    apr = apr/100
    for i in range(numberYears):
        principal = principal * (1 + apr)
    return round(principal)

main()
|
```

```
#Days of Life Calculation V4
```

```
#V4 uses functions for readability, to reduce repetition and to improve
```

```
def main():
```

```
    dobDay, dobMonth, dobYear = getUsersBirthDate()
```

```
    todayDay, todayMonth, todayYear = gettoday()
```

```
    daysOfLife = daysFirstYear(dobDay, dobMonth, dobYear)\
```

```
                + daysWholeYears(dobYear, todayYear)\
```

```
                + daysCurrentYear(todayDay, todayMonth, todayYear)
```

```
    print('You have been alive for', daysOfLife, 'days')
```

```
def daysFirstYear(d,m,y):
```

```
    daysOfMonths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

```
    if leapYear(y):
```

```
        daysOfMonths[1] = 29
```

```
    days = daysOfMonths[m-1]-d
```

```
    for month in daysOfMonths[m:]:
```

```
        days += month
```

```
    return days
```

```
def daysWholeYears(y1,y2):
```

```
    days = (y2-y1-1)*365
```

```
    for year in range(y1,y2+1):
```

```
        if leapYear(year):
```

```
            days += 1
```

```
    return days
```

```
def daysCurrentYear(d,m,y):
```

```
    daysOfMonths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

```
    if leapYear(y):
```

```
        daysOfMonths[1] = 29
```

```
    days = d
```

```
    for month in daysOfMonths[:m-1]:
```

```
        days += month
```

```
    return days
```

```
def leapYear(year):
```

```
    if year%400==0 or (year%400 != 0 and year%100 != 0 and year%4 == 0):
```

```
        return True
```

```
    else:
```

```
        return
```