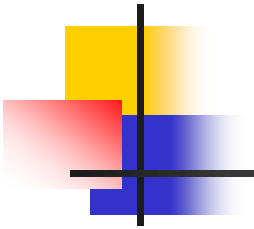


Introduction to Programming

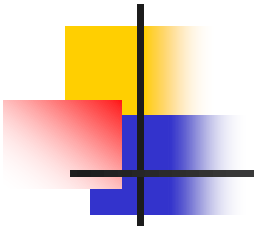
Department of Computer Science and Information Systems

Lecturers: Tingting Han and Steve Maybank

sjmaybank@dcs.bbk.ac.uk

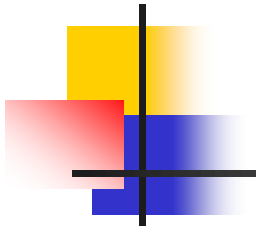
Autumn 2019 and Spring 2020

Week 9: Functions



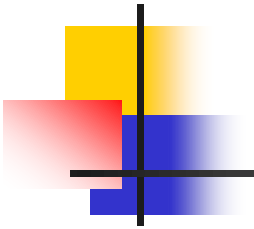
Exercise 2: Vowels

- Input a string, print out the characters in a vertical line, then count the number of lowercase vowels in the string.



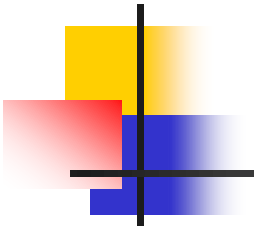
Exercise 2: code

```
s = input("Enter a string:")
n = 0

for letter in s :
    print(letter)
    if letter == "a" or letter == "e" or letter == "i" or letter == "o" or letter == "u"  :
        n = n+1

print("The number of vowels is", n)
```



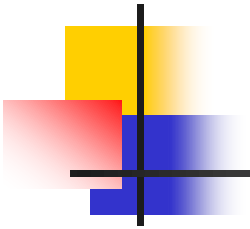
Exercise 2: code

```
s = input("Enter a string:")
n = 0

for letter in s :
    print(letter)
    if letter == "a" or letter == "e" :
        n = n+1
    elif letter == "i" or letter == "o" or letter == "u" :
        n = n+1

print("The number of vowels is", n)
```



Exercise 2: code

```
s = input("Enter a string:")
n = 0

for letter in s :
    print(letter)
    if letter in "aeiou" :
        n = n+1

print("The number of vowels is", n)
```



Exercise 3: number properties

- Write a program to input a non-empty list of strictly positive integers from the keyboard.
- The end of the input is indicated by 0.
- The program then prints out the following numbers.

  The average

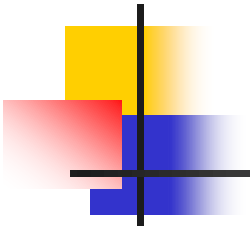
  The smallest of the values

  The largest of the values

  The range

Exercise 3: code(1)

```
number = int(input("Enter a strictly positive integer (0 to finish): "))
count = 1              # number of inputs
total = number         # total of the inputs
mx = number            # maximum value
mn = number            # minimum value

while number > 0 :
    number = int(input("Enter a strictly positive integer (0 to finish): "))
    if number != 0 :
        count = count + 1
        total = total + number
        mx = max(mx, number)
        mn = min(mn, number)

print("The average value is", total/count)
```



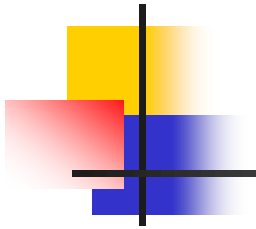
Exercise 3: code(2)

```
print("The average value is", total/count)
print("The smallest value is", mn)
print("The largest value is", mx)
print("The range is", mx-mn+1)
```

Functions are Tools

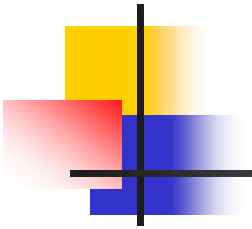
Name	What it does	Input	Output

To use a tool, we don't need to know how it is implemented.
To build a tool, we do need to know how it's implemented.



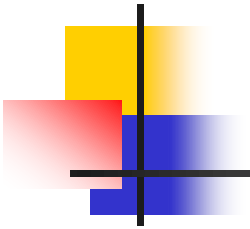
Functions

- A function is a sequence of instructions with a name.
- When a function is called, the instructions are executed.
- When the execution of the instructions is complete, the function may return a value to the calling program.



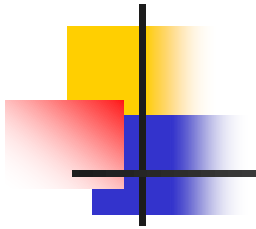
Example

```
price = round(6.8275, 2)

# The function round is called with arguments 6.8275 and 2.

# round returns the number 6.83, which becomes the value of price.

# The computations within round are hidden from the calling program.
```



Function Definition

```
def cubeVolume(sideLength) :    # function header
    volume = sideLength**3      # function body
    return volume               # function body

# name of function: cubeVolume
# Name of parameter variable: sideLength
# def and return are reserved words
# return exits the function and returns the result
```



Compile Time Error

```
print(cubeVolume(10))                #when function is used/called

def cubeVolume(sideLength) :         #when function is created/defined
    volume = sideLength**3
    return volume

# The function cubeVolume is called before it is known to the program
```

Calling a Function from Within a Function

```
def main() :                                    #main() is defined
    result = cubeVolume(2)
    print("A cube with side length 2 has volume", result)
# The definition of cubeVolume is not required when main is defined


def cubeVolume(sideLength) :                    #cubeVolume() is defined
    volume = sideLength**3
    return volume

main()                                          #main() is called, cubeVolume() is called
# The definition of cubeVolume is required when main is called
```

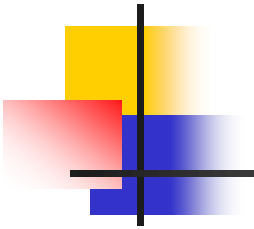
Compile Time Error

```
def main() :                                #main() is defined
    result = cubeVolume(2)
    print("A cube with side length 2 has volume", result)
# The definition of cubeVolume is not required when main is defined


main()          #main() is called, cubeVolume() is called
# The definition of cubeVolume is required when main is called


def cubeVolume(sideLength) : #cubeVolume() is defined
    volume = sideLength**3
    return volume
```

Function Comments

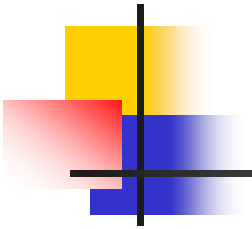
```
##
#  Computes the volume of a cube.
#  @param sideLength the length of a side of the cube
#  @return the volume of the cube
#
def cubeVolume(sideLength) :
   volume = sideLength**3
   return volume
```

Function comments explain the purpose of the function and the meaning of the parameter variables and the return value.



Parameter Passing

- When a function is called, variables are created for receiving the function's arguments.
- **These variables** are called parameter variables or formal parameters.
- **The values** supplied to a function when it is called are the arguments or the actual parameters.



Example

```
def cubeVolume(sideLength) :
    volume = sideLength**3
    return volume

result = cubeVolume(2)
```

The parameter variable sideLength is created when cubeVolume is called

sideLength is initialised with the value 2

The expression sideLength**3 is evaluated, giving 8.

The value 8 is assigned to the variable volume

The function returns. All of its variables are removed. The return value 8 is assigned to the variable result.

Multiple Function Calls

```
def cubeVolume(sideLength) :
    volume = sideLength**3
    return volume

result1 = cubeVolume(2)
result2 = cubeVolume(10)
```

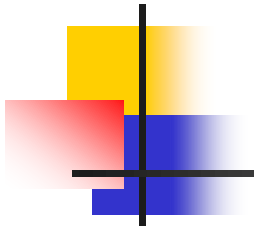
The variables sideLength and volume used in the
calculation of result1 are discarded.

#New variables are created for the calculation of result2.

Test

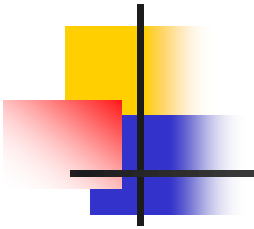
```
def f(x) :
   return g(x) + sqrt(h(x))

def g(x) :
   return 4 * h(x)

def h(x) :
   return x * x + k(x) - 1

def k(x) :
   return 2 * (x + 1)
```

```
# Evaluate f(2) and g(h(2))
```



Cases

```
def cubeVolume(sideLength) :
    if sideLength < 0 :          # deal with the exceptional case
        return 0
    else :
        return sideLength**3     # then deal with the usual case

# Alternative definition
def cubeVolume(sideLength) :
    if sideLength < 0 :
        return 0
    return sideLength**3
```



Branches of a Function

A branch of a function consists of a sequence of instructions that
are carried out when the function is evaluated

This function has two branches, one for sideLength < 0 and
one for sideLength ≥ 0.

```
def cubeVolume(sideLength) :
    if sideLength < 0 :
        return 0
    else :
        return sideLength**3
```

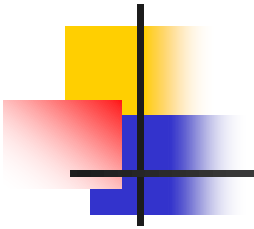
Branches and Return Values

If a function includes return, then every branch should return a value

```
def cubeVolume(sideLength) :
    if sideLength ≥ 0 :
        return sideLength**3
```

Error, no return value for sideLength < 0.
The compiler does not report the error.

```
v = cubeVolume(-1)
```
returns a special value None

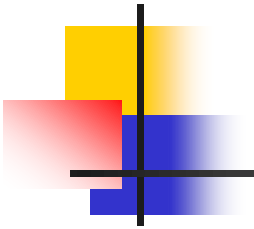


Scope

- The scope of a variable is the part of the program in which it can be accessed.
- A local variable is a variable defined in a function. The scope extends from the line in which it is defined to the end of the function.

```
def main() :
    sum = 0
    for i in range(11)
        square = i * i
        sum = sum + square
    print(square, sum)
```

How many local variables are there? What is the scope of each?

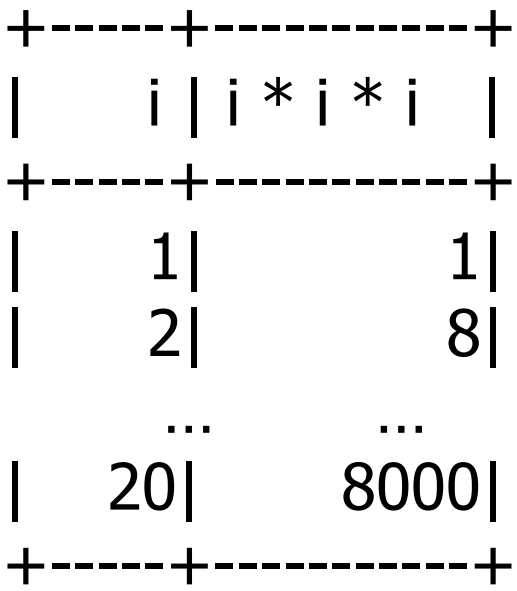


Scope

- The scope of a variable is the part of the program in which it can be accessed.
- A local variable is a variable defined in a function. The scope extends from the line in which it is defined to the end of the function.

```
def main() :
    sum = 0                          # first line in the scope of the local variable sum
    for i in range(11) :             # first line in the scope of the local variable i
        square = i * i               # first line in the scope of the local variable square
        sum = sum + square
    print(square, sum)               # last line in the scope of sum, i, square

# Note: main() has no return value
```

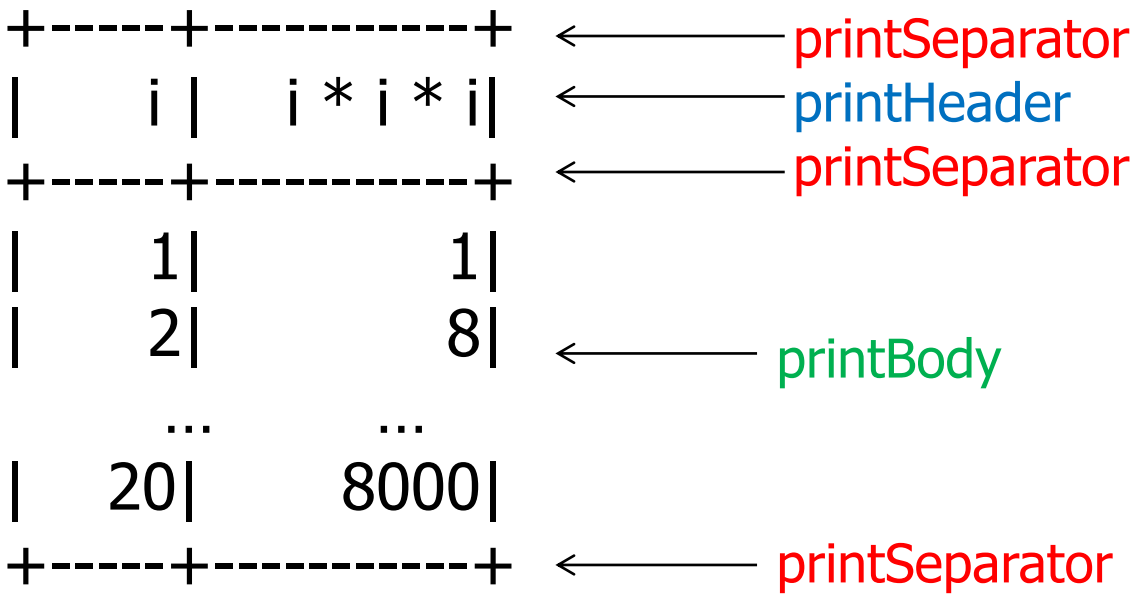
Stepwise Refinement



i	i * i * i
1	1
2	8
…	…
20	8000

Divide the task of printing this table into a sequence of simpler tasks. (PFE, Ch. 5.7, self check 30)

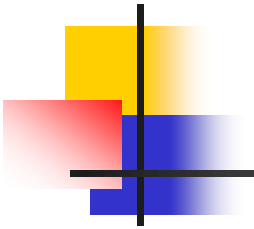
Solution


+-----+----------+
| i | i * i * i|
+-----+----------+
| 1| 1|
| 2| 8|
... ...
| 20| 8000|
+-----+----------+
printSeparator
printHeader
printSeparator
printBody
printSeparator

Solution

```
+-----+----------+
|    i|     i*i*i|
+-----+----------+
|    1|         1|
|    2|         8|
|  ...|       ...|
|   20|      8000|
+-----+----------+
```

```
def printSeparator():
        separator = '+' + '-' * 5 + '+' + '-' * 11 + '+'
        print(separator)

def printHeader():
        header ='|' + '%5s' % 'i' + '|' + '%11s' % 'i*i*i' + '|'
        print(header)

def printBody():
        for i in range(1,21):
                string = '|' + '%5d' % i + '|' + '%11d' % i**3 + '|'
                print(string)

def main():
        printSeparator()
        printHeader()
        printSeparator()
        printBody()
        printSeparator()
```



Example

Write a function

```
def repeat(string, n, delim) :
```

that returns `string` repeated `n` times, separated by the string `delim`. For example

```
repeat("ho", 3, ", ")
```

returns "ho, ho, ho"
(not "ho, ho, ho, " !)

Solution

```
def repeat(string, n, delim) :
    if n <= 0:
        return "n should be greater than 0"          #error message
    else:
        s = string
        for i in range(1,n):
            s = s + delim + string
        return s

def main():
    strg = input("Please enter the string to be repeated:")
    num = int(input("Please enter the number of times to be repeated:"))
    deliminator = input("Please enter the deliminator to separate the strings:")
    print(repeat(strg, num, deliminator))

main()
```

Alternative Solution

```
def repeat(string, n, delim) :
    if n <= 0:
        return "n should be greater than 0"          #error message
    else:
        s = string
        s = s + (delim + string) * (n - 1)
        # Alternatively, we can write        ß
        # s += (delim + string) * (n - 1)
        return s

def main():
    strg = input("Please enter the string to be repeated:")
    num = int(input("Please enter the number of times to be repeated:"))
    deliminator = input("Please enter the deliminator to separate the strings:")
    print(repeat(strg, num, deliminator))

main()
```