6: Procedures & Functions

About repetitive code

- Repetition solves a lot of problems about writing same code over and over again
- Still, there are cases when repetition alone is not enough
- Typical example is a case, where the same code needs to be executed in different parts of the program

Subprograms & programs

- A *subprogram* is a portion of code in a larger program
- Subprogram performs a *specific task* and is relatively independent of main program and other subprograms

Why subprograms?

- To avoid typing same code several times; one subprogram can be easily *called* several times during the execution

- To separate different actions in program from each other

Parameterization

- Subprograms also support **parameterization**
- This allows executing the same code with different input values in different parts of the program without copying and pasting code blocks

Subprograms (cont.)

- Subprograms can be divided into three categories:
- Procedures
- Function
- Methods

Procedures, functions & methods

- **Procedure** is a subprogram with no return value (or an ”empty” return value)
- **Function** is a subprogram which returns a value, like `len(x)`
- **Method** is a subprogram that is connected to an object; methods are also procedures or functions. For example `string.count(x)`

Subprograms in Python

- The syntax for defining a subprogram in Python:

```
def subProgramName(parameter list):
    subprogram body
```

- The parameters are optional; however, the parentheses around them are always required.

Subprogram execution

- The code inside a subprogram is only executed when the subprogram is called
- Same subprogram can be called various times from the "main program" or from other subprograms.

Calling subprograms in Python

- To call a subprogram in Python, we use the subprogram's name:

```
def myProcedure():
  print "Hello"

myProcedure()
```

- Note, that in Python the subprograms should be defined at the beginning of the code to call them from the "main" program.

Calling subprograms in Python (cont.)

- The subprograms can be called from the Python Shell.

- Note, that before calling the subprograms, you must execute the program containing them (in IDLE press **F5** or select *Run* → *Run module*).

What are parameters?

- Parameters are used to give input values to subprograms
- Parameters are subprogram's **local variables**.
- The variable values are defined by calling the subprogram
- Calling the subprogram gives **initial values** to parameter variables

Parameters

- There can be any number of parameters (0…n) in parameter list
- The parameters are separated with a comma character
- The subprogram call needs to have the exactly same amount (and same types) of parameters than the subprogram definition

Parameters and arguments

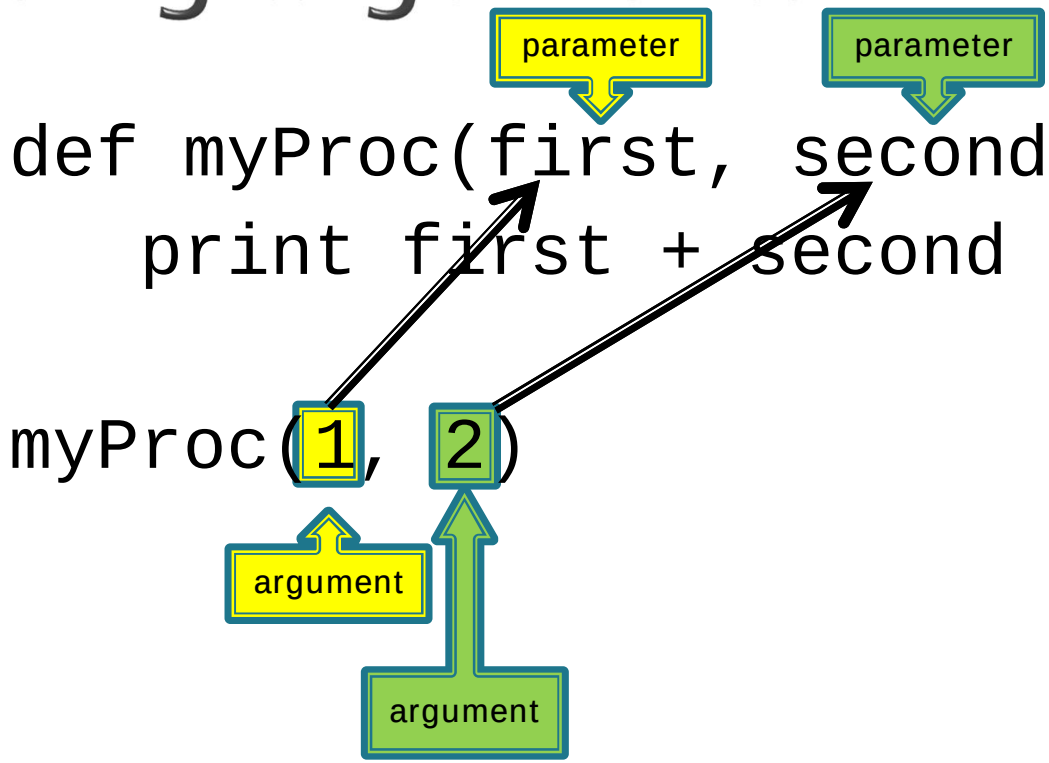
- **Parameter** is a variable name defined in the subprogram definition
- **Argument** is the actual parameter value passed to the subprogram
- When the subprogram is called, the **argument values are assigned into parameter variables**.

Passing arguments

```
def myProc(first, second):
    print first + second

myProc(1, 2)
```

Passing arguments



Example

- A procedure that outputs the square of a number passed to it:

```
def outputSquare(number):
  print number ** 2
```

- To display the squares for numbers 4 and 6:

```
outputSquare(4)
outputSquare(6)
```

Example 2

- Procedure that outputs a middle letter of a string:

```
def outputMiddle(s):
    print s[len(s) / 2]

outputMiddle("abcde") # output c
s = "abc"
outputMiddle(s) # output b
outputMiddle(s + "defg") # output d
outputMiddle(s * 3) # output b
```

Example 3

- Procedure that displays the sum of the arguments:

```
def sum(a, b):
    print a + b

sum(2,4) # output 6
sum(2, 3 + 4) # output 9
sum("ab", "cd") # output abcd
```

Example 4:

- Procedure that displays the smallest of three arguments:

```
def outputSmallest(n1, n2, n3):
    if n1 < n2 and n1 < n3:
        print n1
    elif n2 < n3:
        print n2
    else:
        print n3

outputSmallest(4,7,11) # output 4
outputSmallest(2 + 4, 3 * 3, 18 / 3) # ??
```

Variable scope

- The variables defined inside subprograms body (including parameter variables) are *not visible* outside the subprogram!
- Hence, variables like that are called subprogram's *local variables.*
- Different subprograms and the main program can have variables with identical names; they don't however share the type or the value.

Example

```
def outputValue(n):
    n = n + 1 #affects local variable only
    print n # outputs 5 (4 + 1)

n = 4
outputValue(n)
print n # outputs 4
```

Example 2

```
def increaseAndOutput(n):
    n = n + 1
    print n # outputs n + 1

def multiplyAndOutput(n):
    n = n * 3
    print n # outputs n * 3

n = 3
increaseAndOutput(n) # outputs 4
multiplyAndOutput(n) # outputs 9
print n # output what?
```

Functions

- Functions are subprograms that **return a value**.
- The return value is an object with type and value.
- Return value is similar to evaluation value of expressions: if you need to re-use it later, it needs to be saved into variable.

Returning values in Python

- To return a value from a function, the **return** keyword is used:

```
def myFunction():
    return 47

print myFunction() → outputs 47
a = myFunction() + 3
print a → outputs 50
```

Returning values.. (2)

- The return value "replaces" the function call in the expression:

```
def myFunc():
    return 10

print 2 * myFunc() + 1
```

Returning values.. (2)

- The return value "replaces" the function call in the expression:

```
def myFunc():
    return 10

print 2 * myFunc() + 1
```

Example

- A function that returns the average of three values passed as function arguments:

```
def average(a, b, c):
    sum = a + b + c
    return sum / 3

print average(4,5,9) → Ouputs 6
a = average(1,2,6)
print a → Outputs 3
```

Example 2

- The parameters can be of different types:

```
def multiply(s, n):
    return s * n

print multiply("abc",3) → Outputs abcabcabc
print multiply(4,4) → Outputs 16
```

Returning multiple values

- In Python, it is possible to return more than one value (or a *tuple)* from the function, using syntax

  return value1, value2, ...

- When calling a function like this, all return values must be referenced:

  var1, var2, ... = myFunc()

- However, this is usually not recommended, as a *single function should perform a single task*.

Example

- Function that splits the string into two parts:

```
def splitString(str):
    middle = len(str) / 2
    return str[:middle], str[middle:]

a, b = splitString("abcdef")
print a # outputs abc
print b # outputs def
```

Nested calls

- Subprograms can call other subprograms:

```
def sum(x,y,z):
    return x + y + z

def average(a,b,c):
    return sum(a,b,c) / 3

print average(2,4,6) → outputs 4
```

Infinite loops

- Again, it is possible to create an infinite loop by calling subprograms from each other:

```
def aaa():
    bbb()

def bbb():
    aaa()

aaa()
```

Question

- What would the following program output?

```
def process(a,b):
    return a-b

print process(5, process(2,1))
```

Question 2

- What about the following program?

```
def fact(n):
    if n <= 1:
        return 1
    return n * fact(n -1)

print fact(4)
```

Using subprograms

- When writing a program to solve a specified task, you should:
  - Isolate subproblems or individual tasks
  - → Write a function or a procedure to solve each of these subproblems
  - In the "main program", call the appropriate subprograms in correct order and with correct arguments.

Example: Hangman

- Let’s take a look at a simple Hangman game, and how it can be divided into subprograms
- In hangman game the uses tries to guess a random word one letter at a time
- If the guess is wrong, a picture of a figure hanging is advanced one step. With enough wrong guesses the game is lost.

Example (cont.)

- First, isolate the subtasks involved in the game
- Note, that each subtask does not necessarily require it's own subprogram, as Python provides various built-in functions.

Example (cont)

- Subtasks involved:

1. Picking up a random word
2. Creating the word ”template” for word to be built.
3. Querying for a character
4. Finding out if character is in the word
5. Inserting character into correct places in the built word
6. Drawing the hanging figure
7. Main program for repeating the steps

Example (cont.)

- The signatures of the subprograms to define:

```
def getRandomWord():
    pass

def getBuiltWord(correctWord):
    pass

def queryAnswer():
    pass

def charInWord(word, character):
    pass

def insertChar(correctWord, builtWord, character):
    pass

def drawHangingMan(incorrectGuesses):
    pass
```

Example (cont)

- First step: pick a random word

```
# Returns a random word
def getRandomWord():
    words = "python program keyboard mouse procedure function subprogram"

    # Following picks a random word from string
    # The mechanism is discussed in more detail later in this course
    word = choice(words.split())

    return word
```

Example (cont.)

- Second step: creating the ”template” for guessed word

```
# Returns the word to be built, i.e. a word consisting of
# correct number of hyphens
def getBuiltWord(correctWord):
    return "-" * len(correctWord)
```

Example (cont.)

- Third step: Query the user for a single character

```
# Queries the user for a single character and returns it
def queryAnswer():
    c = ""
    while len(c) != 1:
        c = raw_input("Give a character: ")
        if len(c) != 1:
            print "Give a single character!"
    return c
```

Example (cont.)

- Fourth step: Find out if the character can be found in the word

```
# Returns true, if the given character can be found
# in given word
def charInWord(word, character):
    return word.find(character) > -1
```

Example (cont.)

- Fifth step: insert the given character into correct places in the built word

```
# Inserts given character into correct places in the word
def insertChar(correctWord, builtWord, character):
    # Iterate through the words
    for i in range(len(correctWord)):
        # check if is char to be inserted
        # and not yet inserted
        if correctWord[i] == character and builtWord[i] == "-":
            # replace with char in built word
            builtWord = builtWord[0:i] + character + builtWord[i + 1:]

    return builtWord
```

Example (cont.)

- Sixth step: draw the hanging figure in the current state

```
# Draws the hanging man based on the number of incorrect guesses
# To keep the program simpler, only four alternatives are provided
def drawHangingMan(incorrectGuesses):
    if incorrectGuesses >= 1:
        print " +--- "
        print " |"
        print " o"
    if incorrectGuesses >= 2:
        print "/|\\"
    if incorrectGuesses >= 3:
        print " |"
    if incorrectGuesses == 4:
        print "/ \\"
```

Example (cont.)

- Now, we just need to implement the game logic in the main program by utilizing the subprograms defined
- This requires a loop that goes on until the word is guessed or the hanging figure is completed.

Example (cont.)

```
# Main program
correctWord = getRandomWord()
builtWord = getBuiltWord(correctWord)
incorrectGuesses = 0

while incorrectGuesses < 4 and builtWord != correctWord:
    # output current word and query for letter
    print builtWord
    c = queryAnswer()
    if charInWord(correctWord, c):
        builtWord = insertChar(correctWord, builtWord, c)
    else:
        incorrectGuesses = incorrectGuesses + 1
    drawHangingMan(incorrectGuesses)
    print "\n"

print "The word was", correctWord
if builtWord == correctWord:
    print "You won!"
else:
    print "You lost!"
```