ICT 133
Structured Programming

Seminar 3

Topics

- Definite loop
- Indefinite loops
  - interactive loop
  - sentinel loop.
- Nested loop structures
- Function basics

Control Structures

- Sequence (seminar 1)
  Each statement executes once, from top

- Decision – branching, selection (seminar 2)
  Each statement executes 0 time or once

- **Iteration – loop, iteration, repetition (seminar 3)**
  Each statement executes 0 or more times

Problem

- To display 5 consecutive numbers

```
x = 11
print( x )
x = x + 1
print( x )
x = x + 1
print( x )
x = x + 1
print( x )
x = x + 1
print( x )
```

or

```
x = 11
if x <= 15:
    print( x, end = " " )
    x = x + 1
```

- if is selection (seminar 2)
- if statement executes 0 time or once

`for` Loop

- Executes 0 or more times
  - depends on number of values

Syntax:

```
for <var> in range(start, end, increment):
      <body>
```

- `range` generates values from start to end -1 using increment
- <var>: loop variable takes one value each time for an iteration/loop

`for` loop

```
x = 11
if x <= 15:
    print( x, end = " " )
    x = x + 1
```

```
for x in range(1, 16):
    print( x, end = " " )
```

- range(11, 16)
  generates 11, 12, 13, 14, 15
  using default increment 1

`range` function

- `range(start, stop, step)`

```
for x in range(1, 16):
    print( x, end = " ")
```

```
range(5)                   0 1 3 3 4
range(1, 5+1)              1 2 3 4 5
range(0, 10, 2)            0 2 4 6 8
range(10, 2, -2)           10 8 6 4
range(10, 0, 1)
```

`for` Loop example

```
def main():
    n = int(input("How many numbers do you have? "))
    if n <= 0:
        print("\nThe number {} is invalid".format(n))
    else:
        sum = 0.0
        for i in range(n):
            x = float(input("Enter a number >> "))
            sum = sum + x
        print("\nThe average of the numbers is",
  sum/n)
```

Another Example

- Multiplication of 2 numbers can be done by addition
- E.g. 2 x 3 is 3 + 3 (add 3, 2 times)

```
sum, n, m = 0, 5, 6 # multiply 5 x 6
for i in range(n):
    sum += m
print("{} * {} = {}".format(n, m, sum))
```

`for` Loop – Another use

- The characters in a str is ordered in a sequence

  Syntax:

  ```
  for <var> in <sequence>:
     <body>
  ```

  Example:

  ```
  for ch in "Spam!":
       print (ch, end=" ")
  ```

- `for` loops are definite loops

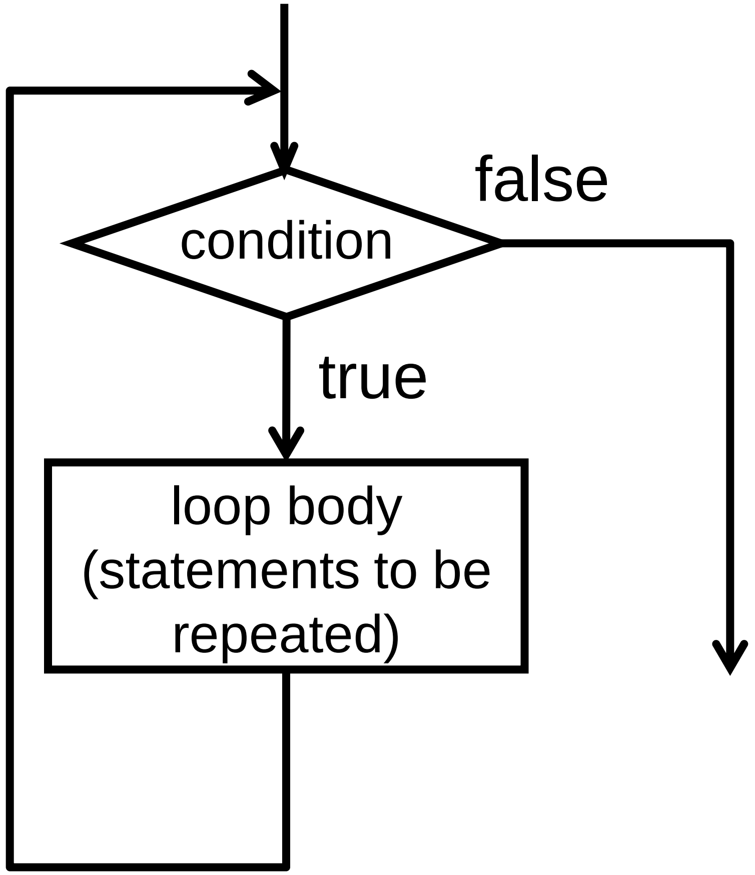
Limitation of Definite Loops

- Can’t use a definite loop unless we know the number of iterations ahead of time.
- Sometimes we can’t know how many iterations we need, e.g., how many numbers to compute the average.

`while` loop

Syntax:

while *condition*:
    statements



Escape route

```
x = 11
while x <= 15:
    print(x, end = ' ')
    x = x + 1
```

There must be a statement that eventually makes the condition false

`while` Loop

- `while` loop that prints 0 to 10

```
i = 0
while i <= 10:
    print(i)
    i = i + 1
```

- `for` loop that prints 0 to 10

```
for i in range(11):
    print(i)
```

Infinite loop

- A common source of program error

(A)

```
i = 0
while i <= 10:
    print(i)
```

(B)

```
i = 5
while i >= 0:
    print(i)
    i = i + 1
```

What is wrong with code (A) and code (B)?

Break statement

- Break statement allows execution to exit the body of a loop.

```
i = 0
while True:
    if i > 10: break
    print(i)
    i = i + 1
```

Interactive Loop

Using string indexing (moredata[0]) allows us to accept “y”, “yes”, “yeah” to continue the loop

```
def main():
    sum, count = 0.0, 0
    moredata = "yes"
    while moredata[0] == "y":
        x = float(input("Enter a number >> "))
        sum = sum + x
        count = count + 1
        moredata = input("Do you have more numbers (yes or no)? ")
    print("\nThe average of the numbers is", sum/count)
```

Interactive Loop with Break

```
def main():
    sum, count = 0.0, 0
    moredata = "yes"
    while True:
        if moredata[0] != "y": break
        x = float(input("Enter a number >> "))
        sum = sum + x
        count = count + 1
        moredata = input("Do you have more numbers (yes
or no)? ")
    print("|nThe average of the numbers is", sum / count)
```

Sentinel Loop

- A *sentinel loop* continues to loop until a special value is reached.
- This special value is called the *sentinel*.
- The sentinel must be distinguishable from the data since it is not processed as part of the data.

Sentinel Loop

We assume that there is no test score below 0, so a negative number is the sentinel

```
def main():
    sum, count = 0.0, 0
    x = float(input("Enter a number (negative to quit) >> "))
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = float(input("Enter a number (negative to quit) >> "))
    print("\nThe average of the numbers is", sum/count)
```

Sentinel Loop with Break

```
def main():
    sum, count = 0.0, 0
    while True:
        x = float(input( "Enter a number (negative to quit) >> "))
        if x < 0: break
        sum = sum + x
        count = count + 1
    print( "|nThe average of the numbers is", sum / count)
```

Application – Guessing Game

- To guess the value of a dice
- Only 3 tries
- Dice value revealed after 3 tries
- Initially, the dice value is hardcoded as 4.

Application – Guessing Game

```
diceValue, tries = 4, 1
while tries <= 3:
    guess = int(input("Try {}. Enter guess: ".format(tries)))
    if diceValue == guess:
        print("You got it!")
        break
    print("Incorrect")
    tries += 1
if  tries > 3:
    print("Sorry, value is {}".format(diceValue))
```

for-else

```
diceValue= 4
for tries in range(3):
    guess = int(input("Try {}. Enter guess: ".format(tries)))
    if diceValue == guess:
        print("You got it!")
        break
    print("Incorrect")
else:
    print("Sorry, value is {}".format(diceValue))
```

Generate random integer

- random.randint(start, end) generates a value between start and end, inclusive of start and end

- To generate a random dice value

```
from random import randint
diceValue = randint(1, 6)
```

Other random functions –

https://docs.python.org/3/library/random.html

Nested Loops

Putting one loop inside another

```
for  i in range (1,  3):
    for j in 'abc':   # loop through j = 'a , 'b', 'c'
        print(i, j)
```

```
1  a
1  b
1  c
2  a
2  b
2  c
```

Application – Extend Guessing game

- After each game, prompt if the user wishes to continue

E.g.

```
Try No 1. Enter guess: 4
Incorrect.
Try No 2. Enter guess: 5
You got it!
Continue? (y/n): y
Try No 1. Enter guess: ….
```

Application – Extend Guessing game

```
playAgain = 'y'
while playAgain[0].lower() == 'y':
    diceValue, tries = randint(1, 6), 1
    while tries <= 3:
        …
        tries += 1

    if  tries > 3:
        print("Sorry, value is {}".format(diceValue))
    playAgain = input("Continue? y/n: ")

print("End game")
```

Application – Extend Guessing game

```
from random import randint
playAgain = 'y'
while playAgain[0].lower() == 'y':
    diceValue, tries = randint(1, 6), 1
    while tries <= 3:
        guess = int(input("Try {}. Enter guess: ".format(tries)))
        if diceValue == guess:
            print("You got it!")
            break
        print("Incorrect")
        tries += 1
    if  tries > 3:
        print("Sorry, value is {}".format(diceValue))
    playAgain = input("Continue? y/n:  ")
print("End game")
```

Functions

- Types of functions:
  - `main()`
  - Built-in Python functions (`print`)
  - Functions from the standard libraries (`random.randint`)

Functions reduce code duplication and make programs more easily understood and maintained.

Defining and Calling

- Function definition

```
def main():
    print("Happy birthday to you!" )
    print("Happy birthday to you!" )
    print("Happy birthday, dear Fred...")
    print("Happy birthday to you!")
```

A sequence of statements is given a name

- Function call

```
>>> main()
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred...
Happy birthday to you!
```

To get the statements in the function to execute

Function call in Function Definition

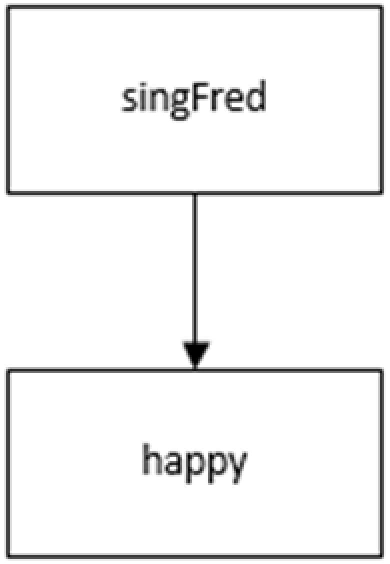
```
def happy():
    print("Happy birthday to you!")

def singFred():
    happy()
    happy()
    print("Happy birthday, dear Fred...")
    happy()

>>> singFred()
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Fred...
Happy birthday to you!
```

Function call in a function definition

Structure Chart



Parameter

What if it's Lucy's birthday? We need to write a new singLucy function!

- A generic function

```
def sing(person):
    happy()
    happy()
    print("Happy birthday, dear", person + ".")
    happy()
```

  - uses parameter person.

A (formal) *parameter* is a variable that is initialized when the function is called.

Function Call with Parameter

Formal parameter

↓

```
def sing(person):
    happy()
    happy()
    print("Happy birthday,
dear", person + ".")
    happy()
```

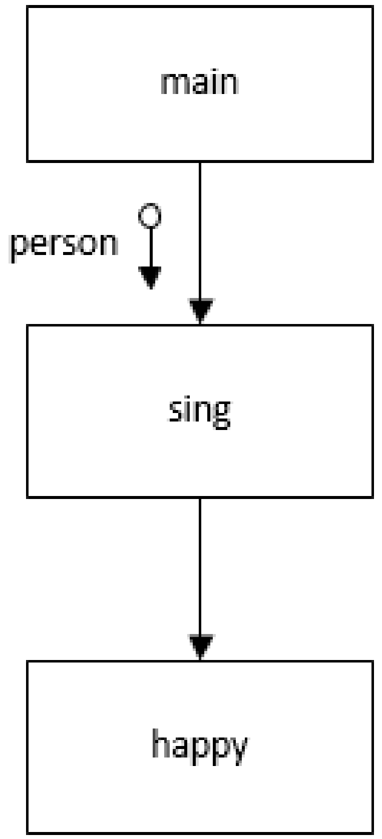
```
def main():
    name = input("Enter name: ")
    sing(name)
```

↑

Actual parameter

```
>>> main()
Enter name: Lucy
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lucy.
Happy birthday to you!
```

Structure Chart with Parameter



Function Definition

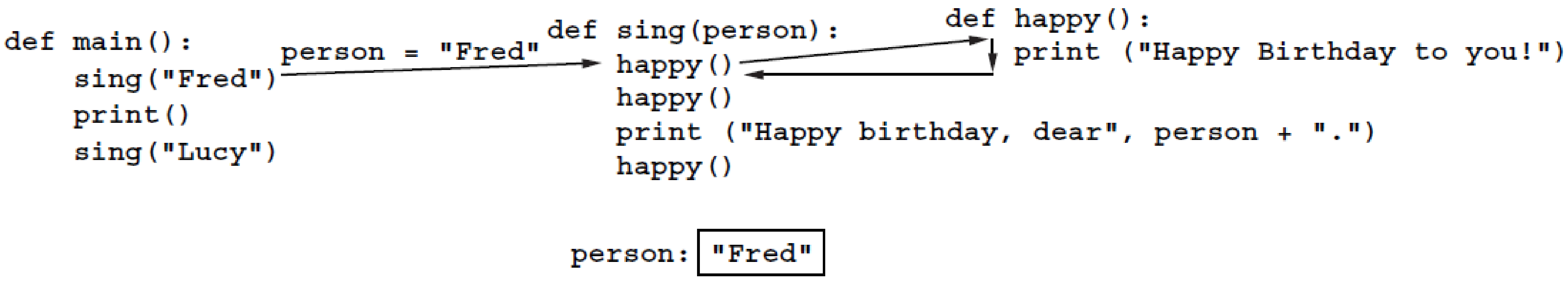
```
def <name>(<formal-parameters>):
    <body>
```

- name must be an identifier
- There can be 0, 1 or more formal-parameters, separated by commas.

4-step process in function call

- Calling program suspends execution
- Formal parameters get assigned the values of the actual parameters
- The body of the called function is executed
- Calling program continues at the point just after the function call

4-step process in function call



```
def main():
    sing("Fred")
    print()
    sing("Lucy")
```

person = "Fred"

```
def sing(person):
    happy()
    happy()
    print ("Happy birthday, dear", person + ".")
    happy()
```

```
def happy():
    print ("Happy Birthday to you!")
```

person: "Fred"

Source: Python Programming, 3/e 39

Function That Returns Values

```
discrim = math.sqrt(b*b - 4*a*c)
```

- The value `b*b - 4*a*c` is the *actual* parameter of `math.sqrt`
- `sqrt` function *returns* a value which is the square root of the parameter

`return` statement

Example:

```
def square(x):
    return x*x
```

- The value(s) in the `return` statement are returned to the caller
- `return` exits the function

Returning several Values

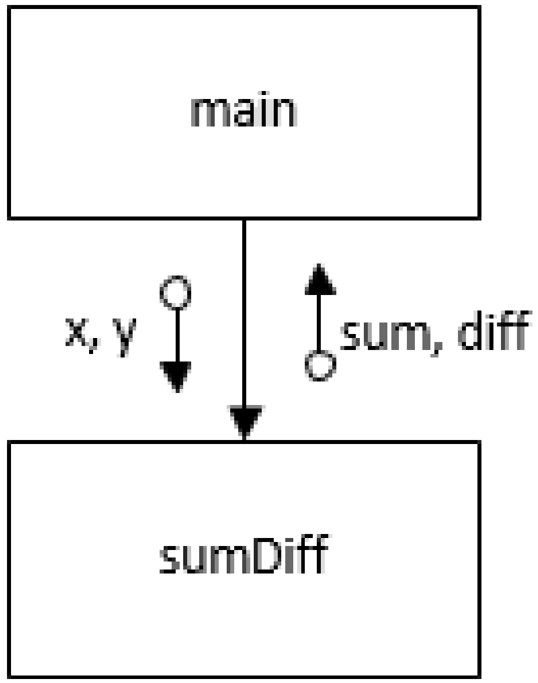
```
def sumDiff(x, y):
    sum = x + y
    diff = x – y
    return sum, diff
```

Function call:

```
    s, d = sumDiff(num1, num2)
```

- Functions without a `return` statement return `None`.

Structure Chart with Returned Values


main
x, y
sum, diff
sumDiff