15-110 Principles of Computing – F21

Lecture 8:
For Loops, Range function

Teacher:
Gianni A. Di Caro

Repeating actions

Function that returns the sum of the first 10 numbers?

```
def sum_first():
        ?
```

```
def sum_first():
    s = 0
    s = s + 1
    s = s + 2
    s = s + 3
    s = s + 4
    s = s + 5
    s = s + 6
    s = s + 7
    s = s + 8
    s = s + 9
    s = s + 10
    return s
```

Repeating the same action with different inputs for 10 times

$$s = s + i, \text{ with } i = 1, 2, \ldots 10$$

Can we summarize this repetitive action in one instruction?

```
def sum_first():
    s = 0
    for i in 1,2,3,4,5,6,7,8,9,10:
        s = s + i
    return s
```

Repeating actions: `for` loops

o General form of a
   `for` loop construct:

```
for variable in sequence_of_things:
    do_something
```

scope of loop:
indent!

sequence_of_things: sequence of numbers, characters, strings, ...

variable *loop index*: at each <u>iteration</u>, **the variable is set to the value of the next item in the sequence**

```
def sum_first():
    s = 0
    for i in 1,2,3,4,5,6,7,8,9,10:
        s = s + i
    return s
```

```
def sum_first():
    s = 0
    for i in 1,2,3,4,5,6,7,8,9,10:
        s = s + i
    return s
```

o How many times will the instruction(s) in the body of the loop (`s = s + i`) be executed?

➢ **Number of iterations**: 10

✓ The number of iterations is (pre)**defined**

❖ `while` loops for undefined number of iterations

4

Loop enrolling

```python
def sum_first():
  s = 0
  for i in 1,2,3,4,5,6,7,8,9,10:
    s = s + i
  return s
```

Operationally the same as:

Equivalent to:

```python
def sum_first():
  s = 0
  s = s + 1
  s = s + 2
  s = s + 3
  s = s + 4
  s = s + 5
  s = s + 6
  s = s + 7
  s = s + 8
  s = s + 9
  s = s + 10
  return s
```
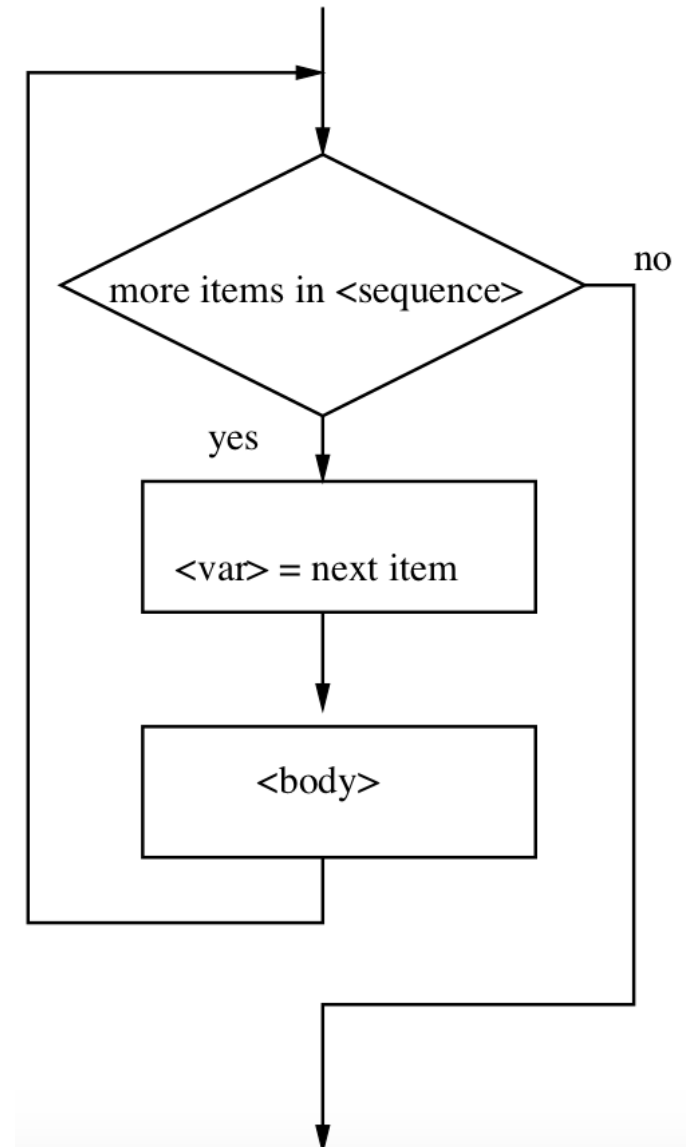
```python
def sum_first():
  s = 0
  i = 1
  s = s + i
  i = 2
  s = s + i
  i = 3
  s = s + i
  i = 4
  s = s + i
  i = 5
  s = s + i
  i = 6
  s = s + i
  i = 7
  s = s + i
  i = 8
  s = s + i
  i = 9
  s = s + i
  i = 10
  s = s + i
  return s
```

Definite loops: `for` construct

✓ Repeat a set of <u>actions</u> a **defined number of times** (at *most*)

✓ Each time the action *can* be executed on a <u>different input parameter</u>

Sequences of *things*?

➢ Can we have sequences of **anything**? → Yes!

```
def print_numbers ():
    for i in 1, 2.5, 99.1, -2, 0:
        print(i)
```

```
def print_mix():
    for i in True, 2.5, 99.1, False:
        print(i)
```

- We will learn how to write sequences in different forms

For the time being, we can use sequences as lists of *things* separated by commas

Sequences of integer numbers: `range()` function

➢ If we want to repeat something n times, where n is passed as an input, we need to have a compact way to **automatically generate a sequences of** n **numbers**

```
def sum_first(n):
          ?
```

o Write a function that prints out *Happy birthday* n times, where n is a variable input

```
def happy_bday(n):
    for i in range(n):
        print('Happy birthday!')
```

✓ Note that here we are not even using the loop index variable in the body of the for loop, we only need to have a counter for repeating things

<u>Equivalent to:</u>

```
def happy_bday(n):
    for i in 0, 1, 2, ..., n-1:
        print('Happy birthday!')
```

← We can't run this code!!!

8

Sequences of integer numbers: `range()` function

➤ `range(n)` is a function that *returns* a sequence: `0, 1, 2, ..., n-1`

```
def sum_first(n):
            ?
```

```
def sum_first(n):
    s = 0
    for i in range(n+1):
        s = s + i
    return s
```

✓ In this case, we are using the loop index variable `i` for the computations in the body of the for loop

Note : `range()` returns an **object that produces a sequence of integers (a *generator*)**

range(start,end,step)

range(start, end, step) returns an object that produces a sequence of integers from start (inclusive) to end (exclusive) by step.

```python
s = 0
for i in range(1, 10, 2):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(-2, 10, 2):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(0, 10, 5):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(-5, 11, 5):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

Short forms: `range(end)`, start from 0, steps of 1

`range(end)` returns an object that produces a <u>sequence of integers</u> from 0 to end (exclusive) by steps of 1

```python
s = 0
for i in range(5):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(0, 5, 1):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(10):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

Short forms: `range(start,end)`, steps of 1

`range(start, end)` returns an object that produces a <u>sequence of integers</u> from start (inclusive) to end (exclusive) by steps of 1

```python
s = 0
for i in range(10, 15):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(-10, 1):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(10, 15, 1):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

range(start, end, back_steps), start > end

If start > end, step must be **negative**! (stepping backward)

```python
s = 0
for i in range(10, 5, -1):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(20, 12, -2):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(0, -5, -1):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

```python
s = 0
for i in range(10, 5):
    print('Loop variable:', i)
    s = s + i
print('Sum:', s)
```

For loops for summing up

Write the function `sum_all (r_min, r_max)` that sums up all the integer numbers between the integer values `r_min` and `r_max` (included).

```python
def sum_all(r_min, r_max):
    s = 0
    for i in range(r_min, r_max+1):
        s = s + i
    return s
```

For loops for summing up the odds numbers

Write the function `sum_odds(r_min, r_max)` that sums up all the integer odd numbers between the integer values `r_min` and `r_max` (included).

```python
def sum_odds(r_min, r_max):
    s = 0
    for i in range(r_min, r_max+1):
        if i % 2 != 0:
            s = s + i
    return s
```

If we know that `r_min` is an odd number:

```python
def sum_odds(r_min, r_max):
    s = 0
    for i in range(r_min, r_max+1, 2):
        s = s + i
    return s
```

Watch out: the loop index variable

```
a = 3
b = 10
s = 0
for i in range(a, b, 2):
    s = s + i
print(s)
print(i) ?????
```

➢ **Loop index variable**:

✓ The variable created for the for loop <u>doesn't disappear after the loop is done</u>

✓ It will contain the <u>last value</u> used in the `for` loop

```
a = 3
b = 10
s = 0
for i in range(a, b+1, 2):
    s = s + i
print(s)
print(i)
```

```
a = 3
b = 10
s = 0
for i in range(a, b+2, 2):
    s = s + i
print(s)
print(i)
```

Watch out: the loop index variable

```python
a = 3
b = 10
s = 0
for i in range(a, b, 2):
    s = s + i
    i = 2 * i

print(s)
print(i) ?????
```

➢ **Loop index variable**:

✓ It *can* be changed inside the loop

✓ Don't do it!

For loops for summing up: Alternating sum

The **alternating sum** of a sequence of numbers $a_1, a_2, a_3, a_4.., a_n$ is defined as: $a_1 - a_2 + a_3, -a_4+..\pm a_n$

Write the function `alternatingSum(n)` that returns the alternating sum of all integer numbers between 1 and n, inclusive.

```python
def alternatingSum(n):

    s_pos = 0
    for i in range(2, n+1, 2):
        s_pos = s_pos + i

    s_neg = 0
    for j in range(1, n+1, 2):
        s_neg = s_neg + j

    return s_pos - s_neg
```

```python
def alternatingSum(n):
    s = 0
    for i in range(1, n+1):
        if i % 2 == 0:
            s = s + i
        else:
            s = s - i
    return s
```

For loops for summing up after selection

Write the function `sum_if_divisible(a, b)` that returns the sum of all the integer numbers between the integer values a and b (included) that can be divided by 2 or by 3, or by both (e.g., 9 can be divided by 3, 6 can be divided by both, 5 can't be divided neither by 2 or by 3).

```python
def sum_if_divisible(a, b):
    s = 0
    for v in range(a, b+1):
        if (v % 2 == 0) or (v % 3 == 0):
            s = s + v
    return s
```

For loops for summing up: Fibonacci's sequence

❖ The **Fibonacci Sequence** is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

➤ The next number in the sequence is found by <u>adding up the two numbers before it</u>:
- $a_1 = 2 \leftarrow$ (1+1)
- $a_2 = 3 \leftarrow$ (1+2)
- $a_3 = 5 \leftarrow$ (2+3)
- …

✓ The first two numbers, a_1, a_2 are given: different two initial numbers generate different (sub)sequences

Write the function `Fibonacci(a1, a2, n)` that computes and prints out all the integer numbers in the sequence up to sequence index `n` (included) based on the first two numbers in the sequence, `a1`, `a2`.

For loops for the perfect numbers

In mathematics, a **perfect number** is an integer for which the sum of all its own positive divisors (excluding itself) is equal to the number itself. For example, the number 6 is perfect, because 1+2+3 is equal to 6. Other perfect numbers are 28, 496, and 8,128.

Implement the function `isPerfect(n)` that returns `True` if `n` is a perfect number, or `False` otherwise.

```python
def isPerfect(n):
    s = 0
    for i in range(1, n):
        if n % i == 0:
            s = s + i
    if s == n:
        return True
    else:
        return False
```

For loops for summing up, a bit more general scenario

Write the function `sum_odds_general(r_min, r_max)` that sums up all the integer odd numbers between the values `r_min` and `r_max` (included).
`r_min` and `r_max` is can be any numbers and aren't necessarily integers but $r\_min \leq r\_max$

```python
import math

def sum_odds_general(r_min, r_max):
    s = 0
    r_min = math.floor(r_min)
    r_max = math.ceil(r_max)
    print(r_min, r_max)

    if r_min % 2 == 0:
        start = r_min + 1
    else:
        start = r_min

    for i in range(start, r_max+1, 2):
        s = s + i
    return s
```