Lecture 3

Iteration

Burkay Genç, Ahmet Selman Bozkır, and Selma Dilek

15/03/2023

PREVIOUS LECTURE

- strings
- branching - if/elif/else
- indentation
- while loops

TODAY

- while loops
- for loops
- loop exercises

WHILE LOOP

- the while loop is used to do **something** as long as a **condition** holds

```
while <condition>:
   do_something
   do_something_else
   do_something_more
```

- You must make sure that the loop **terminates**
  - Otherwise, you get an **infinite loop**

PRINTING TO k

- Print the numbers from 1 to a given k (inclusive)

PRINTING TO k

- Print the numbers from 1 to a given k (inclusive)

```
k = 5
i = 1
while i <= k:
   print(i)
   i = i + 1
```

```
## 1
## 2
## 3
## 4
## 5
```

PRINTING ODD NUMBERS

- Print the odd numbers from 1 to a given k (inclusive)

PRINTING ODD NUMBERS

- Print the odd numbers from 1 to a given k (inclusive)

```
k = 15
i = 1
while i <= k:
   if i % 2 == 1:
      print(i)
   i = i + 1
```

```
## 1
## 3
## 5
## 7
## 9
## 11
## 13
## 15
```

PRINTING ODD NUMBERS

- Print the odd numbers from 1 to a given k (inclusive)
- **Alternative (shorter) solution**

```
k = 15
i = 1
while i <= k:
   print(i)
   i = i + 2
```

```
## 1
## 3
## 5
## 7
## 9
## 11
## 13
## 15
```

FACTORISATION

- Print all factors of a given number

FACTORISATION

- Print all factors of a given number

```
number = 28
i = 1
while i <= number:
   if number % i == 0:
      print(i)
   i = i + 1
```

```
## 1
## 2
## 4
## 7
## 14
## 28
```

PRIME NUMBERS

- Print whether a given number is prime

PRIME NUMBERS

- Print whether a given number is prime

```
number = 28
i = 2
isPrime = True

while i < number and isPrime:
   if number % i == 0:
      isPrime = False
   i = i + 1

if isPrime:
   print(number, "is prime.")
else:
   print(number, "is not prime.")
```

```
## 28 is not prime.
```

PRIME NUMBERS

- Print whether a given number is prime

```
number = 29
i = 2
isPrime = True

while i < number and isPrime:
   if number % i == 0:
      isPrime = False
   i = i + 1

if isPrime:
   print(number, "is prime.")
else:
   print(number, "is not prime.")
```

```
## 29 is prime.
```

FOR LOOP

FOR LOOP

- The `for` loop allows iterating in a fixed sequence

```
for <variable> in <sequence of values>:
   do_something
```

- each time through the loop, `<variable>` takes a value from the `<sequence of values>`
- values are selected according to the given sequence

RANGE

- The `range` function provides a sequence of integer values
- `range(5)` means numbers from `0` (inclusive) to `5` (exclusive)
  - `0, 1, 2, 3, 4`
- We can use this with the for loop:

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

```
## 0
## 1
## 2
## 3
## 4
```

- Read the code!
  - For i in numbers from 0 to 5, print the value of i.

RANGE

- You can fine tune range to provide different sequences of numbers
- Normally it has 3 *arguments*
  - An argument is a parameter input to a function
- `range(start, stop, step)`
- when we write `range(5)` it means `range(stop = 5)`
  - `start` defaults to 0
  - `step` defaults to 1

RANGE

- `range(4, 10, 1)` means
  - `start = 4`
  - `stop = 10`
  - `step = 1`
- `4, 5, 6, 7, 8, 9`

```
for i in range(4, 10, 1):
   print(i)
```

```
## 4
## 5
## 6
## 7
## 8
## 9
```

RANGE

- `range(2, 8, 2)` means
  - `start = 2`
  - `stop = 8`
  - `step = 2`
- `2, 4, 6`
- `stop` is **exclusive**

```
for i in range(2, 8, 2):
   print(i)
```

```
## 2
## 4
## 6
```

RANGE

- `range(10, 0, -1)` means
  - `start = 10`
  - `stop = 0`
  - `step = -1`
- `10, 9, 8, 7, 6, 5, 4, 3, 2, 1`

```
for i in range(10, 0, -1):
   print(i)
```

```
## 10
## 9
## 8
## 7
## 6
## 5
## 4
## 3
## 2
## 1
```

BREAK

- The `break` statement is used to **immediately** exit a loop
  - `while` and `for` loops both work
- Skips remaining expressions in code block
- Exits only **innermost** loop!

```
while <condition_1>:        # Loop 1
   while <condition_2>:     # Loop 2
      <expression_a>
      break                 # Exits loop 2
      <expression_b>
   <expression_c>
```

EXAMPLE

- Stop printing when a number **divisible by 7** is found

```
for i in range(10, 20, 1):
   print(i)
   if i % 7 == 0:
      break
```

```
## 10
## 11
## 12
## 13
## 14
```

PRIME NUMBERS

- **Revisited!** Print whether a given number is prime
- We now know the `for` loop, as well as `break`
  - Can we use them to produce a *shorter* algorithm?
  - *Faster*?
  - *Cleaner*?
  - More *readable*?
- A good programmer's job is not to produce code
  - A good programmer's job is to produce short, fast, clean, readable code.

PRIME NUMBERS

- **Revisited!** Print whether a given number is prime

```
number = 29

for i in range(2, number):     # step defaults to 1
   if number % i == 0:
      print(number, "is not prime.")
      break

if i == (number - 1):
   print(number, "is prime.")
```

```
## 29 is prime.
```

- Can you be faster?
  - Do you really need `range(2, number)`?

FOR vs WHILE

`for` loops

- know number of iterations
- can end early via `break`
- uses a counter
- can rewrite a `for` loop using a `while` loop

`while` loops

- unbounded number of iterations
- can end early via break
- can use a counter but must initialize before loop and increment it inside loop
- may not be able to rewrite a `while` loop using a `for` loop

LITERAL SEQUENCES

- We can also provide a sequence literally

```
for i in [1, 3, 9, 24]:
   print(i)
```

```
## 1
## 3
## 9
## 24
```

```
for i in ["a", "b", "c", "d"]:
   print(i)
```

```
## a
## b
## c
## d
```

```
for i in ["a", 1, "c", True]:
   print(type(i))
```

```
## <class 'str'>
## <class 'int'>
## <class 'str'>
## <class 'bool'>
```

EXERCISES

EXERCISE

- Print a string in reverse.

EXERCISE

- Print a string in reverse.

```
name = "burkay genc"
for i in range(len(name) - 1, -1, -1):
   print(name[i], end = "")
```

```
## cneg yakrub
```

EXERCISE

- Check if a substring exists in a string
  - If so, return its position

EXERCISE

- Check if a substring exists in a string
  - If so, return its position

```
s = "Check if a substring exists in a string"
ss = "st"

for i in range(len(s) - len(ss)):
   if s[i:(i + len(ss))] == ss:
      print(i)
```

```
## 14
## 24
## 33
```

EXERCISE

- Given a **random sequence** of numbers, find the *value* of the **maximum number** in the sequence

```
sequence = [4, 7, 1, 0, -8, 12, -3, 1, 4, 11, 9, -2, 10]
```

EXERCISE

- Given a **random sequence** of numbers, find the *value* of the **maximum number** in the sequence

```
sequence = [4, 7, 1, 0, -8, 12, -3, 1, 4, 11, 9, -2, 10]
max = sequence[0]

for i in sequence:
   if i > max:
      max = i
print(max)
```

```
## 12
```

EXERCISE

- Given a **random sequence** of numbers, find the *position* of the **maximum number** in the sequence

```
sequence = [4, 7, 1, 0, -8, 12, -3, 1, 4, 11, 9, -2, 10]
```

EXERCISE

- Given a **random sequence** of numbers, find the *position* of the **maximum number** in the sequence

```
sequence = [4, 7, 1, 0, -8, 12, -3, 1, 4, 11, 9, -2, 10]
max = 0

for i in range(len(sequence)):
   if sequence[i] > sequence[max]:
      max = i
print(max)
```

```
## 5
```

EXERCISE

- Fibonacci sequence
  - `1, 1, 2, 3, 5, 8, 13, 21, ...`
  - $F_i = F_{i-1} + F_{i-2}$
- Print the first `k` Fibonacci numbers

EXERCISE

- Print the first `k` Fibonacci numbers

```
k = 15
F = 1
F_1 = 0
F_2 = 0
i = 1

while i <= k:
   print(F)
   F_2 = F_1
   F_1 = F
   F = F_1 + F_2
   i = i + 1
```

```
## 1
## 1
## 2
## 3
## 5
## 8
## 13
## 21
## 34
## 55
## 89
## 144
## 233
## 377
## 610
```

HOMEWORK EXERCISES

- Make a game of guess
  - The user tries to guess a number stored in the program
  - If the user's guess is higher than the stored value, output `Guess is too high`, and ask for a new guess
  - If the user's guess is lower than the stored value, output `Guess is too low`, and ask for a new guess
  - If the guess is succesfull, output 'You Win!!!' and quit the program.

HOMEWORK EXERCISES

- Write a program that asks for a word from the user
- Then, the program checks whether the word is a **palindrome**
- A palindrome is a string that is equal to its reverse
  - `kayak`
  - `kelek`
  - `abcdcba`

HOMEWORK EXERCISES

- Write a program that asks for a string from the user and then asks for a single character
- Then, the program returns the number of occurances of that character in the given string
- For example,
  - Given the string `Korkma sönmez bu şafaklarda yüzen al sancak` and the character `k`, the program should output 3
  - `K` and `k` are assumed to be different!