Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas

Universidade Federal de Viçosa

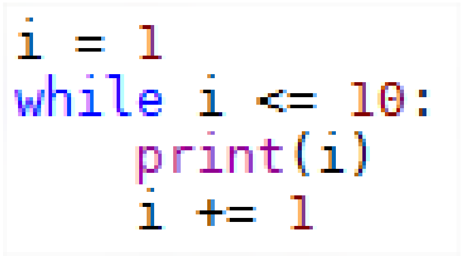
INF 100 – Introduction to Programming

Repetition

command *for*

Loops with *while* command

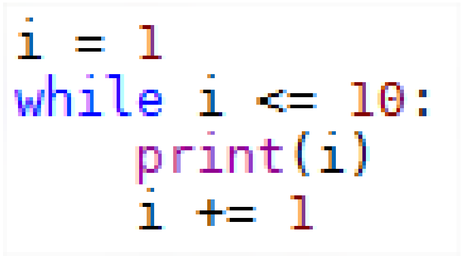
- What is the output for the code below?



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

Loops with *while* command

- What is the output for the code below?

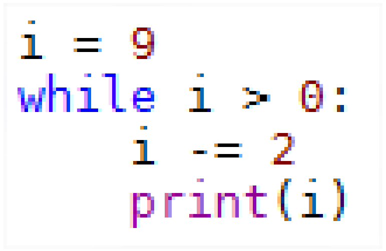


```
i = 1
while i <= 10:
    print(i)
    i += 1
```

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

Loops with *while* command

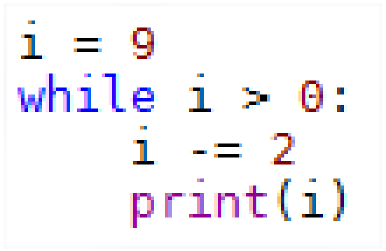
- What is the output for the code below?



```
i = 9
while i > 0:
    i -= 2
    print(i)
```

Loops with *while* command

- What is the output for the code below?

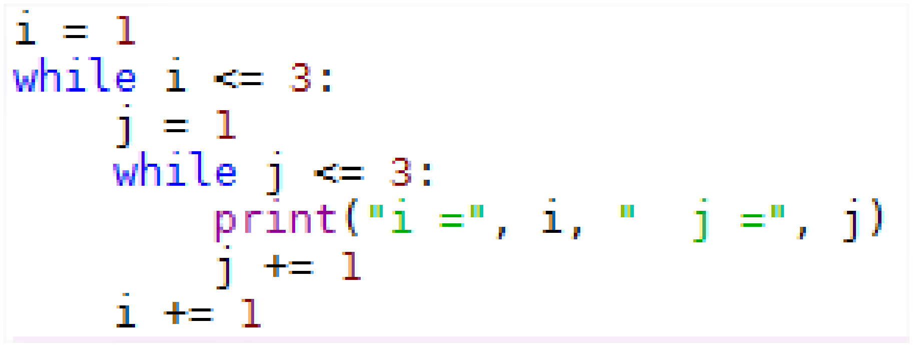


```
i = 9
while i > 0:
    i -= 2
    print(i)
```

```
7
5
3
1
-1
```

Loops with *while* command

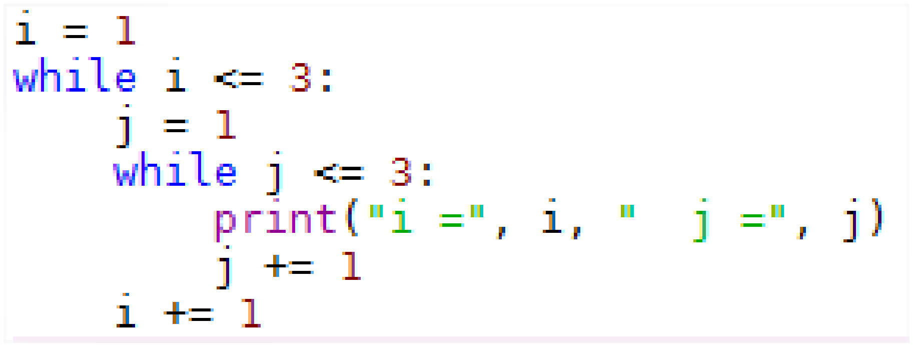
- What is the output for the code below?



```
i = 1
while i <= 3:
    j = 1
    while j <= 3:
        print("i =", i, "  j =", j)
        j += 1
    i += 1
```

Loops with *while* command

- What is the output for the code below?



```python
i = 1
while i <= 3:
    j = 1
    while j <= 3:
        print("i =", i, "  j =", j)
        j += 1
    i += 1
```

```
i = 1   j = 1
i = 1   j = 2
i = 1   j = 3
i = 2   j = 1
i = 2   j = 2
i = 2   j = 3
i = 3   j = 1
i = 3   j = 2
i = 3   j = 3
```

Number of repetitions: known x unknown

```
while True:
    n = int(input("Type the number of the month: "))
    if (n >= 1 and n <= 12):
        break
    print("Month must be between 1 and 12")
```

```
i = 1
while i <= n:
    print(i)
    i += 1
```

Number of repetitions: known x unknown

- Unknown:

```
while True:
    n = int(input("Type the number of the month: "))
    if (n >= 1 and n <= 12):
        break
    print("Month must be between 1 and 12")
```

- Known:

```
i = 1
while i <= n:
    print(i)
    i += 1
```

The command FOR in algorithms

- Cases in which a variable must assume values inside a given range are so common in programs that many programming languages have a special command for it.
- In algorithms, this command is represented by the keyword “FOR”.

The command FOR in algorithms

- General use:

```
for i = initial_value to final_value:
    execute a command
```

- The code above means that the given command will be executed several times.
- In the first iteration, **`i`** has value equal to **`initial_value`**.
- In the second iteration, **`i`** has value equal to **`initial_value`**+1.
- The process continues until the last iteration, when **`i`** has value **`final_value`**.

Using FOR - examples

```
for i = 1 to 5:
    print(i)
```

Using FOR - examples

```
for i = 1 to 5:
    print(i)
```

```
1
2
3
4
5
```

Using FOR - examples

```
for i = 1 to 5:
    print(i)
```

```
1
2
3
4
5
```

```
for j = 3 to 6:
    print(j)
```

Using FOR - examples

```
for i = 1 to 5:
    print(i)
```

```
1
2
3
4
5
```

```
for j = 3 to 6:
    print(j)
```

```
3
4
5
6
```

Variations of the command FOR

- Variations of the command FOR allow increasing or decreasing the control variable by values other than 1 unit:

```
for i = initial_value to final_value, step k:
    execute a command
```

- In the code above, variable **`i`** will start with value equal to **`initial_value`**.
- In the next iteration, **`i`** will be **`initial_value+k`**, then **`initial_value+2k`**, ..., until **`i`** has value **`final_value`**.

Using FOR - examples

```
for i = 0 to 10, step 2:
    print(i)
```

```
0
2
4
6
8
10
```

```
for j = 12 to 3, step -3:
    print(j)
```

```
12
9
6
3
```

Command FOR in Python

- Simple syntax:

```
for i in range(initial,final):
    execute a command
```

- Semantics: variable **`i`** will start with value equal to **`initial`**.
- In the next iteration, **`i`** will be **`initial+1`**, then **`initial+2`**, …
- The repetition stops **BEFORE** **`i`** has value **`final`**.

Command FOR: algorithms x Python

(algorithm)

```
for i = 1 to 5:
    print i
```

(Python)

```
for i in range(1, 6):
    print( i )
```

```
1
2
3
4
5
```

In Python, the loop is interrupted BEFORE the control variable assumes this value

Variations

- Extended syntax:

```
for i in range(initial,final,step):
    execute a command
```

- Semantics: variable **`i`** will start with value equal to **`initial`**.
- In the next iteration, **`i`** will be **`initial+step`**, then **`initial+2*step`**, ...
- The repetition stops BEFORE **`i`** has value **`final`**.

Examples

```
for i in range(0,10,2):
    print(i)
```

Examples

```
for i in range(0,10,2):
    print(i)
```

```
0
2
4
6
8
```

Examples

```
for i in range(0,10,2):
    print(i)
```

```
0
2
4
6
8
```

```
for i in range(0,11,2):
    print(i)
```

Examples

```
for i in range(0,10,2):
    print(i)
```

```
0
2
4
6
8
```

```
for i in range(0,11,2):
    print(i)
```

```
0
2
4
6
8
10
```

Examples

```
for j in range(15,9,-1):
    print(j)
```

Examples

```
for j in range(15,9,-1):
    print(j)
```

```
15
14
13
12
11
10
```

Examples

```
for j in range(15,9,-1):
    print(j)
```

```
15
14
13
12
11
10
```

```
for k in range(15,9,-2):
    print(k)
```

Examples

```
for j in range(15,9,-1):
    print(j)
```

```
15
14
13
12
11
10
```

```
for k in range(15,9,-2):
    print(k)
```

```
15
13
11
```

Processing several values (revisited)

- In a previous lecture, we discussed 2 approaches for reading and processing several values.
- One approach: before start reading the values, the program asks the user the number of values that will be typed.
- Sample execution (calculating average):

```
How many values? 3
Type a value: 7
Type a value: 18
Type a value: 5
Average = 10
```

Python template

```
n = int(input("How many values? "))
i = 1
# other initializations

while i <= n:
    value = float(input("Type a value: "))
    # process value
    i += 1

# process the results
```

Python template (rewritten with FOR)

```
n = int(input("How many values? "))
# other initializations

for i in range (1, n+1):
    value = float(input("Type a value: "))
    # process value

# process the results
```

Average of a set of values

```
while True:
    n = int(input("How many values? "))
    if n > 0:
        break
    print ("Number of values must be positive!")

sum = 0
for i in range (1, n+1):
    value = float(input("Type a value: "))
    sum += value

avg = sum / n
print("Average = ", avg)
```

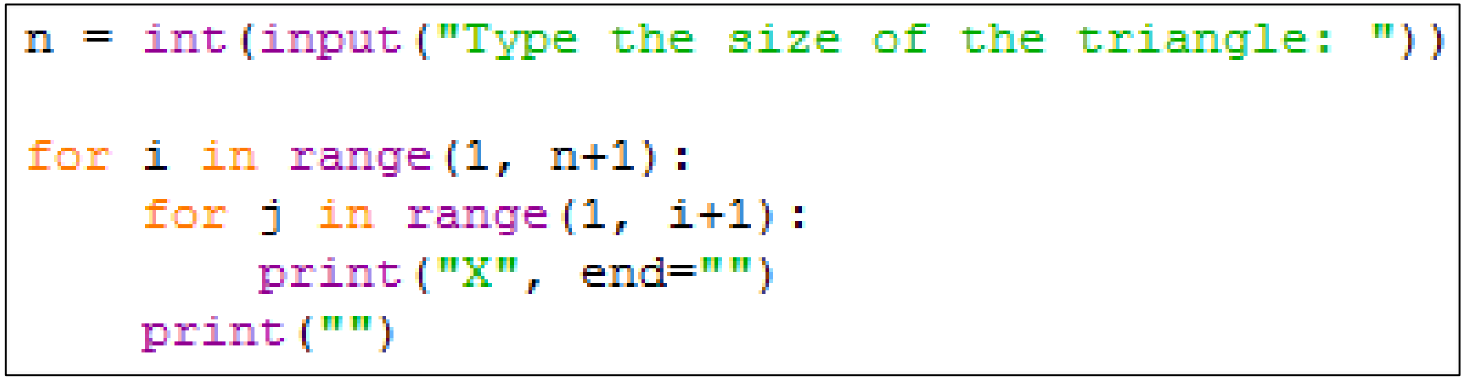
Exercise

- Write two Python programs to draw a triangle rectangle using symbols “X”, following the given examples of execution.
- The user must be requested to type an integer number representing the size of the triangle. Assume that it is a positive number.

Triangle #1 – example of execution

```
type the size of the triangle: 5
x
xx
xxx
xxxx
xxxxx
```

Solution



```
n = int(input("Type the size of the triangle: "))

for i in range(1, n+1):
    for j in range(1, i+1):
        print("X", end="")
    print("")
```

Triangle #2 – example of execution

```
type the size of the triangle: 5
    X
   XX
  XXX
 XXXX
XXXXX
```

Solution

```
n = int(input("Type the size of the triangle: "))

for i in range(1, n+1):
    for j in range(1, n-i+1):
        print(" ", end="")
    for j in range(1, i+1):
        print("X", end="")
    print("")
```

Exercise

- An integer number greater than 1 is **prime** if it has no positive divisors other than 1 and itself.
- Problem: given an integer number, decide whether it is a prime number or not.

Prime? - algorithm

- Read a positive integer number *n*
- Check if *n* is divisible by 2, 3, 4, …, n-1
- If *n* is not divisible by any of the numbers above, it is a prime number; otherwise, it is not a prime number.

Prime? – more detailed algorithm

```
read n, a positive integer number
isPrime ← true
for i = 2 to n-1:
    if n is divisible by i then
        isPrime ← false
if isPrime then
    print n "is prime"
else
    print n "is not prime"
```

Prime? – more detailed algorithm

```
read n, a positive integer number
isPrime ← true
for i = 2 to n-1:
    if n is divisible by i then
        isPrime ← false
if isPrime then
    print n "is prime"
else
    print n "is not prime"
```

Improve it!
Stop the loop when a divisor is found

Prime? – more detailed algorithm

```
read n, a positive integer number
isPrime ← true
for i = 2 to n-1:
    if n is divisible by i then
        isPrime ← false
        stop the loop
if isPrime then
    print n "is prime"
else
    print n "is not prime"
```

Prime? – version in Python

```python
while True:
    n = int(input("Type a positive number: "))
    if n > 0:
        break
    print(n, "is not a positive number")

isPrime = True
for i in range(2,n):
    if n % i == 0:
        isPrime = False
        break
if isPrime:
    print(n, "is a prime number")
else:
    print(n, "is not a prime number because it is divisible by", i)
```

Prime? – improved algorithm

- Read a positive integer number *n*
- Check if *n* is divisible by 2
- Check if *n* is divisible by 3, 5, 7, ..., $\sqrt{n}$
- If *n* is not divisible by any of the numbers above, it is a prime number; otherwise, it is not a prime number.

Prime? – more detailed algorithm

```
read n, a positive integer number
isPrime ← true
if n is greater than 2 and n is divisible by 2 then
    isPrime ← false
else
    for i = 3 to $\sqrt{n}$, step 2:
        if n is divisible by i then
            isPrime ← false
            stop the loop
if isPrime then
    print n “is prime”
else
    print n “is not prime”
```

Prime? – version in Python

```
while True:
    n = int(input("Type a positive number: "))
    if n > 0:
        break
    print(n, "is not a positive number")

isPrime = True
if n > 2 and n % 2 == 0:
    isPrime = false
else:
    limit = int(n**0.5)
    for i in range(3, limit+1, 2):
        if n % i == 0:
            isPrime = False
            break
if isPrime:
    print(n, "is a prime number")
else:
    print(n, "is not a prime number because it is divisible by", i)
```