Introduction to Python Programming

5 – Control Structures

Josef van Genabith (Stefan Thater)
Dept. of Language Science & Technology
Universität des Saarlandes

WS 2022/23

Recap

- What is an expression?
- What is a literal?
- What is a variable?
- What’s the difference between expressions and statements?
- What are assignments?
- What’s the difference between
  - x = x + 1 and
  - x == x + 1 ?

Imperative Programming

- Imperative / procedural programming:
  - First do this, then do this.
- Elements of imperative programs
  - Variables
  - Assignments
  - Expressions
  - **Control Structures:** loops, branches

Statements

- Python program = sequence of statements = sequence of instructions Python can execute
- Seen so far:
  - assignments
  - print (actually, an expression)
- Statement ≈ a step in the underlying algorithm
  - separated by line breaks

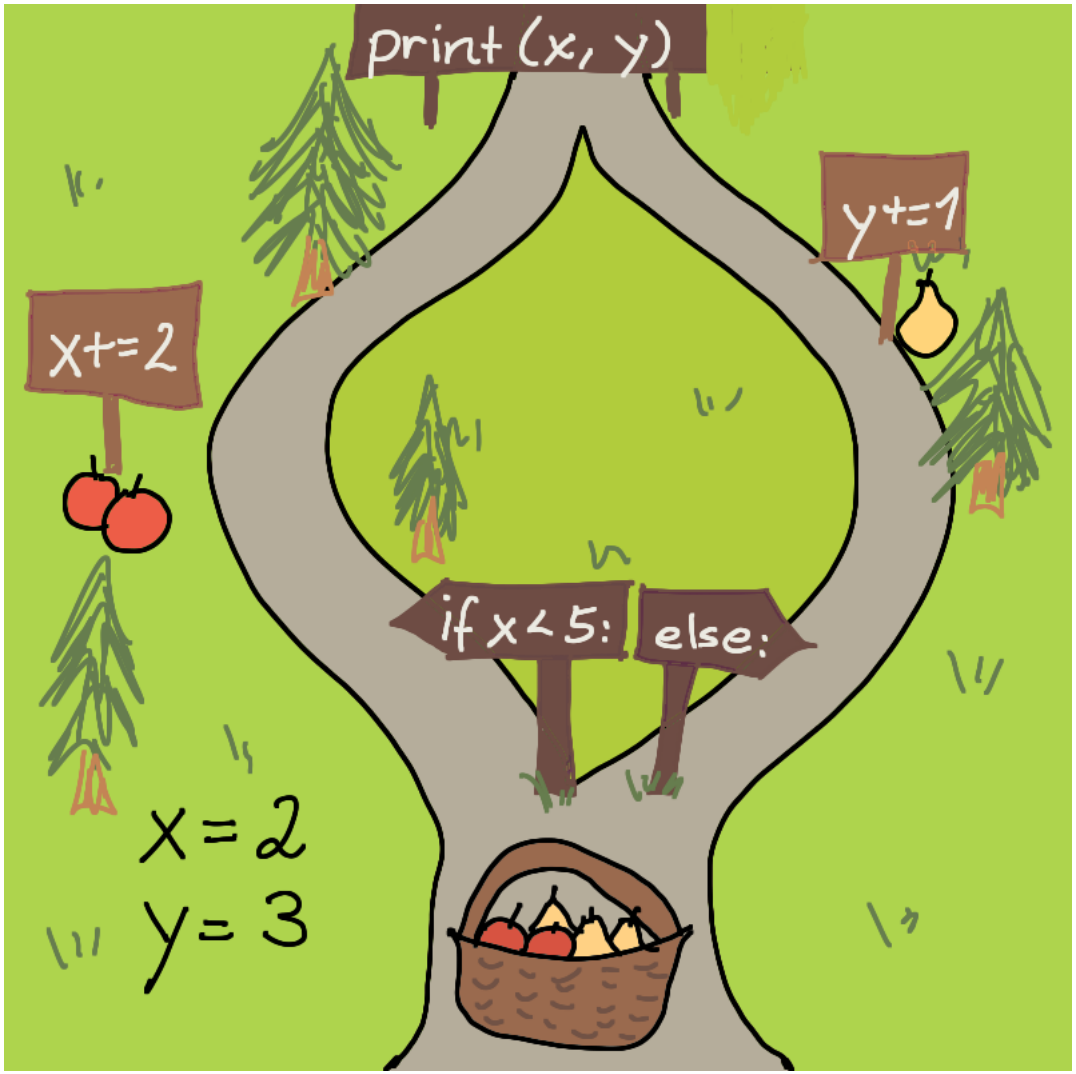
```
a = input("A number, please: ")
b = input("Another number, please: ")
r = a + b
print(a, "+", b, "=", a + b)
```

Control Structures

Sometimes we want to execute a (block of) statement(s)

- repeatedly: `while`, `for` (loops)
- only under certain conditions: `if` (conditionals)

Conditions



Conditions: `if` … `else` …

- if $\texttt{expr}_1$ evaluates to a value that counts as `true`, $\texttt{block}_1$ is executed; otherwise, $\texttt{block}_2$ is executed
  - Values counting as false: `False`, `0`, the empty string (`''`), empty lists, empty sets …
  - All other values count as true.

```
if expr1:
  block1
else:
  block2
```

```
if expr1:
  block1
```

- A block consists of one or more statements; blocks are indicated by indentation.
- The `else`-part is optional

Condition: `if` …

- Read in an integer
- Test whether integer is even or odd

Condition: `if` …

- Read in an integer
- Test whether integer is even or odd

```
# prompt for input
print("Please enter an integer: ")

x = input()     # read the input
num = int(x)    # ensure num is integer

mod = num % 2   # modulo division

if mod == 0:
    print(num,"is even.")

if mod != 0:
    print(num,"is odd.")
```

Condition: `if` …

- Read in an integer
- Test whether integer is even or odd

```
# prompt for input
print("Please enter an integer: ")

x = input()     # read the input
num = int(x)    # ensure num is integer



if num % 2 == 0:
    print(num,"is even.")

if num % 2 != 0:
    print(num,"is odd.")
```

Condition: `if … else …`

- Read in an integer
- Test whether integer is even or odd

```
# prompt for input
print("Please enter an integer: ")

x = input()     # read the input
num = int(x)    # ensure num is integer



if num % 2 == 0:
    print(num,"is even.")
else:
    print(num,"is odd.")
```

Condition: `if` …

- Read in a character
- Test whether the character is a vowel
- Execute the block whose corresponding test-expression evaluates to true.

Conditions: `if … elif … else`

- Execute the first (!) block whose corresponding test-expression evaluates to true.
- `else` is optional.

```
if expr1:
  block1
elif expr2:
  block2
elif ...:
  ...
else:
  blockn
```

Condition: `if … elif … else`

- Read in a character
- Test whether the character is one of the vowels
- Execute the first block whose corresponding test-expression evaluates to true.

```
# prompt for input
print("Please enter a character: ")

x = input()    # read the input
chr = str(x)   # ensure chr is a string

if chr == "a":
    print(chr,"is a vowel.")
elif chr == "e":
    print(chr,"is a vowel.")
elif chr == "i":
    print(chr,"is a vowel.")
elif chr == "o":
    print(chr,"is a vowel.")
elif chr == "u":
    print(chr,"is a vowel.")
else:
    print(chr,"is not a vowel.")
```

Condition: `if` …

- Read in a character
- Test whether the character is a vowel
- Execute the block whose corresponding test-expression evaluates to true.
- Is this a good/correct implementation?

```
# prompt for input
print("Please enter a character: ")

x = input()    # read the input
chr = str(x)   # ensure chr is a string

if chr == "a":
    print(chr,"is a vowel.")

if chr == "e":
    print(chr,"is a vowel.")

if chr == "i":
    print(chr,"is a vowel.")

if chr == "o":
    print(chr,"is a vowel.")

if chr == "u":
    print(chr,"is a vowel.")
else:
    print(chr,"is not a vowel.")
```

Condition: `if` …

- Read in a character
- Test whether the character is a vowel
- Execute the block whose corresponding test-expression evaluates to true.
- Is this a good/correct implementation?
- Think hard about this one …
- What‘s wrong with it?

```
# prompt for input
print("Please enter a character: ")

x = input()    # read the input
chr = str(x)   # ensure chr is a string

if chr == "a":
    print(chr,"is a vowel.")

if chr == "e":
    print(chr,"is a vowel.")

if chr == "i":
    print(chr,"is a vowel.")

if chr == "o":
    print(chr,"is a vowel.")

if chr == "u":
    print(chr,"is a vowel.")
else:
    print(chr,"is not a vowel.")
```

Condition: `if` …

- Read in a character
- Test whether the character is a vowel
- Execute the block whose corresponding test-expression evaluates to true.
- Test it with `chr = b`
- Test it with `chr = a` … what happens?

```
# prompt for input
print("Please enter a character: ")

x = input()    # read the input
chr = str(x)   # ensure chr is a string

if chr == "a":
    print(chr,"is a vowel.")

if chr == "e":
    print(chr,"is a vowel.")

if chr == "i":
    print(chr,"is a vowel.")

if chr == "o":
    print(chr,"is a vowel.")

if chr == "u":
    print(chr,"is a vowel.")
else:
    print(chr,"is not a vowel.")
```

Condition: `if` ...

```
# prompt for input
print("Please enter a character: ")

x = input()    # read the input
chr = str(x)   # ensure chr is a string

if chr == "a":
    print(chr,"is a vowel.")
elif chr == "e":
    print(chr,"is a vowel.")
elif chr == "i":
    print(chr,"is a vowel.")
elif chr == "o":
    print(chr,"is a vowel.")
elif chr == "u":
    print(chr,"is a vowel.")
else:
    print(chr,"is not a vowel.")
```

≠

```
# prompt for input
print("Please enter a character: ")

x = input()    # read the input
chr = str(x)   # ensure chr is a string

if chr == "a":
    print(chr,"is a vowel.")

if chr == "e":
    print(chr,"is a vowel.")

if chr == "i":
    print(chr,"is a vowel.")

if chr == "o":
    print(chr,"is a vowel.")

if chr == "u":
    print(chr,"is a vowel.")
else:
    print(chr,"is not a vowel.")
```

Please explain why ...

Indentation is **IMPORTANT**

- Indentation is used to indicate the structure of the code

```
if a < b:
  if a < c:
    print('foo')
  else:
    print('bar')
```

```
if a < b:
  if a < c:
    print('foo')
else:
  print('bar')
```

Indentation is **IMPORTANT**

- Indentation is used to indicate the structure of the code

```
if a < b:
  if a < c:
    print('foo')
  else:
    print('bar')
```

```
if a < b:
  if a < c:
    print('foo')
else:
  print('bar')
```

```
a = 1
b = 3
c = 2
```

```
a = 4
b = 3
c = 2
```

```
a = 2
b = 3
c = 2
```

Indentation is **IMPORTANT**

- Indentation is used to indicate the structure of the code

```
if a < b:
  if a < c:
    print(a,'<',c)
  else:
    ................
```

```
if a < b:
  if a < c:
    print(a,'<',c)
else:
  ................
```

Indentation is **IMPORTANT**

- Indentation is used to indicate the structure of the code

```
if a < b:
  if a < c:
    print(a,'<',c)
  else:
    print(a,'<',b)
```

```
if a < b:
  if a < c:
    print(a,'<',c)
else:
  print(a,'>=',b)
```

Blocks

- Block = grouping of statements
- instructions of the same block must be indented by the same type of whitespace characters (blanks or tabs)
- Best practice: never mix blanks and tabs; always stick to the same type of whitespace!
- Using an IDE (such as IDLE, PyCharm, Anaconda) makes your life easier.

```
if a < b:
  print("foo")
  a += 1
else:
  print("bar")
  b -= 1
```

Blocks

- Block = grouping of statements
- instructions of the same block must be indented by the same type of whitespace characters (blanks or tabs)
- Best practice: never mix blanks and tabs; always stick to the same type of whitespace!
- Using an IDE (such as IDLE, PyCharm, Anaconda) makes your life easier.

```
if a < b:
  print(a,'<',b)
  a += 1
  print(a)
else:
  print(a,'=>',b)
  a -= 1
  print(a)
```

Exercise

- What are the values of a, b and c after executing the following piece of code?

```
a = b = 2
c = False
if not c:
    if b < a:
        b += 5
        a = b-1
    elif a < b:
        c = True
    else:
        if a+b < 4:
            c = False
        a = 11
        b = 2.2
print(a,  b,  c)
```

Exercise: `if` vs `elif`

```
if x:
  print("Hello")
elif y:
  print("World")
else:
  print("Bye bye")
```

```
if x:
  print("Hello")
if y:
  print("World")
else:
  print("Bye bye")
```

What is printed for

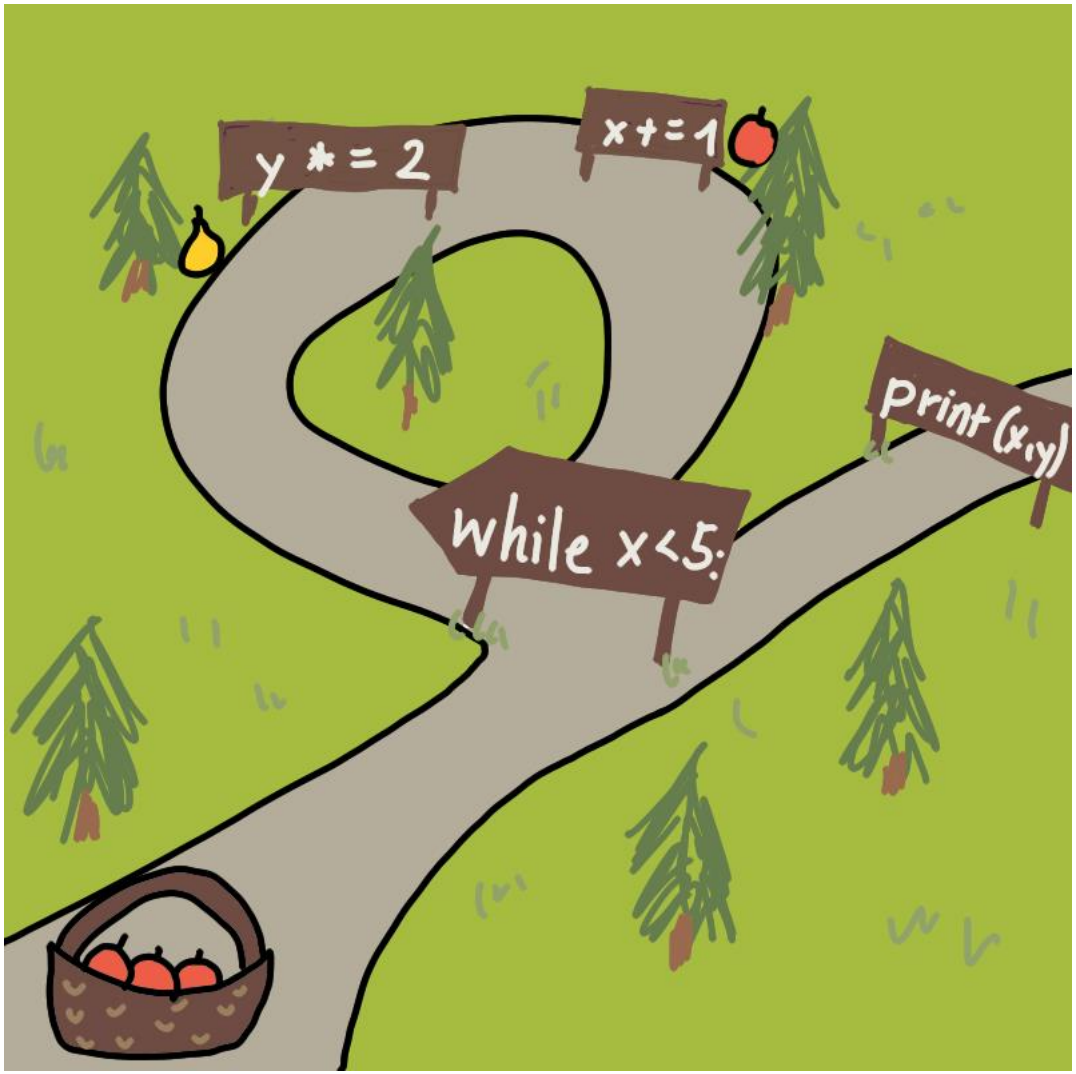
(a) x=True; y=True

(b) x=False; y=True

(c) x=True; y=False

(d) x=False; y=False

Loops: `while`



Loops: `while`

```
while expr:
   block
...
```

- Evaluate `expr`
- If the resulting value is (counts as) false: continue the program after the while-block with the next statement on the same indent level as `while`.
- Otherwise: execute the statements of block. Then go back to the beginning of the while loop: evaluate `expr` again and ...

Exercise

- What is the output of the following program?

```
a = 1
while a < 5:
  print("Hi")
  a = a + 1
```

Exercise

- What is the output of the following program?

```
a = 1
while a < 5:
  print("Hi")
  a = a + 1
  print(a)
```

Exercise

- What is the output of the following program?

```
a = 1
while a < 5:
  print(a)
  print("Hi")
  a = a + 1
  print(a)
```

Exercise

- What is the output of the following program?

```
a = 1
while a < 5:
   print(a)
   print("Hi")
   a = a + 1
   print(a)
print(a)
```

Exercise

- What is the output of the following programs?

```
a = 1
while a < 5:
   print("Hi")
   a = a + 1
```

```
a = 1
while a < 5:
   print("Hi")
   a += 1
```

Exercise

- What is the output of the following programs?

```
a = 1
while a < 5:
   print("Hi")
   a = a + 1
```

```
a = 0
while a < 5:
   print("Hi")
   a = a + 1
```

```
a = 1
while a < 5:
   print("Hi")
   a += 1
```

```
a = 0
while a < 5:
   print("Hi")
   a += 1
```

Exercise

- What is the output of the following program?

```
a = 8
b = 1
while a > 1:
   b += 3
   a = a / 2
print(a, b)
```

Exercise

- What is the output of the following program?

```
a = 1
while a < 5:
  print("Hi")
  a = a - 1
```

Exercise

- Implement a program that computes the factorial of a given number:
  - 1! = 1
  - 2! = 1 * 2 = 2
  - 3! = 1 * 2 * 3 = 6
  - …

```
number = int(input("Please enter a number: "))

# YOUR CODE

print("The factorial of", number, "is", xxx)
```

Exercise

- Implement a program that computes the factorial of a given number:

```
number = int(input("Please enter a number: "))

# YOUR CODE

counter = 1
factorial = 1

while counter <= number:
    factorial = factorial * counter
    counter = counter + 1

print("The factorial of", number, "is", factorial)
```

Exercise

- Implement a program that computes the factorial of a given number:

```
n = int(input("Please enter a number: "))

# YOUR CODE

c = 1
f = 1

while c <= n:
    f = f * c
    c = c + 1

print("The factorial of", n, "is", f)
```

Exercise

- Implement a program that computes the factorial of a given number:

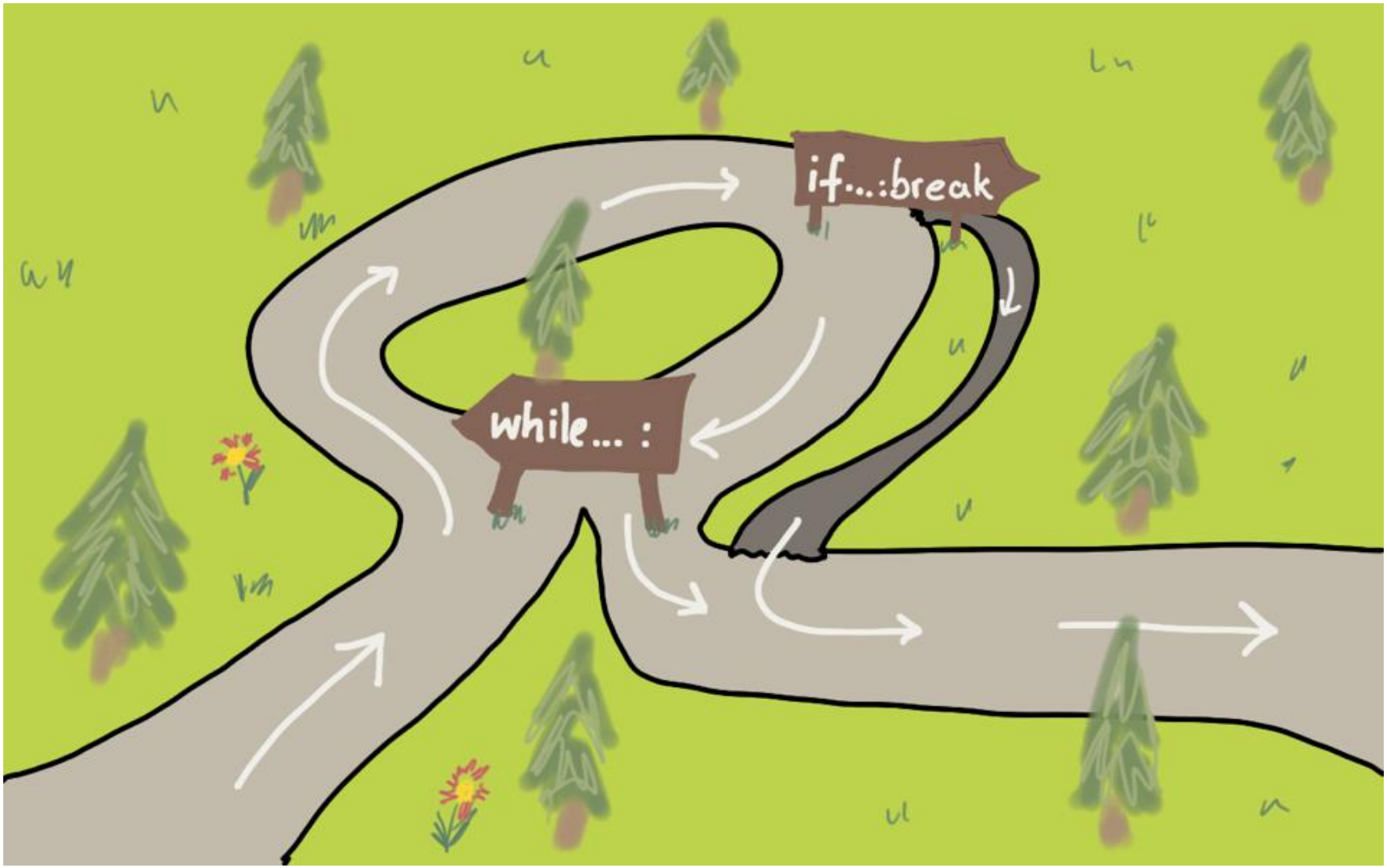
```
n = int(input("Please enter a number: "))

# YOUR CODE

c = 1
f = 1

while c <= n:
    f = *= c
    c = += 1

print("The factorial of", n, "is", f)
```

Exercise

- Implement a program that computes the factorial of a given number:

```
n = int(input("Please enter a number: "))

# YOUR CODE

f = 1

while n >= 1:
    f = *= n
    n = -= 1

print("The factorial of", n, "is", f)
```

The `break` statement



The `continue` statement

`break` and `continue`

- `break` exits the current loop
- `continue` skips the remainder of the current iteration and continues at the beginning: `expr1` is evaluated, and the while-block is executed (if `expr1` evaluates to true).

```
while expr1:
   ...
   if expr2:
      ...
      break
   ...
block
```

```
while expr1:
   ...
   if expr2:
      ...
      continue
   ...
block
```

`for` loops

```
for elt in some_list:
  print(elt)

for i in range(0, 5):
  print(i)
```

- For loops iterate over all elements of list-like values.
- We often want to iterate over all numbers in a certain range:
  - `range(start, end)` creates a corresponding list
  - The end point is not included in the list
  - `range(0, 5) => [0, 1, 2, 3, 4]`
- Actually, `range` evaluates to an iterator, not a list. If you want to get a proper list (e.g., for printing)
  - `x = list(range(0, 5))`

range

- In general: `range(start, end, step)`
  - `range(0, 10, 1) => [0, 1, 2, ..., 9]`
  - `range(0, 10, 2) => [0, 2, 4, 6, 8]`
  - `range(10, 0, -2) => [10, 8, 6, 4, 2]`
- Short forms:
  - `range(0, 10) = range(0, 10, 1)`
  - `range(10) = range(0, 10, 1)`

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

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

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

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

`for` and `while`

```
for elt in some_list:
  print(elt)
```

```
i = 0 # a new variable, not used anywhere else
while i < len(some_list):
  elt = some_list[i]
  print(elt)
  i += 1
```

- Think about `for` loops as shorthands for a `while` loop:
  - ⇒ `break`, `continue` can also be used in for-loops
  - Recommendation: `for` loop often (not always) preferred over `while`

Exercise

- What's the output of the following program?

```
fruits = ["apple", "banana", "melon"]
for i in range(2, 6, 2):
  for fruit in fruits:
    print(str(i) + " " + fruit + "s")
```

```
fruits = ["apple", "banana", "melon"]
for i in range(2, 6, 1):
  for fruit in fruits:
    print(str(i) + " " + fruit + "s")
```

Exercise

- What's the output of the following program?

```
fruits = ["apple", "banana", "melon"]
for i in range(2, 6):
  for fruit in fruits:
    print(str(i) + " " + fruit + "s")
```

```
fruits = ["apple", "banana", "melon"]
for fruit in fruits:
  for i in range(2, 6):
    print(str(i) + " " + fruit + "s")
```

Exercise

- Implement a program that computes the sum over a list of numbers:

```
# compute the sum of a list of numbers

listOfNumbers = [2,4,1,8,3,5]
```

Exercise

- Implement a program that computes the sum over a list of numbers:

```
# compute the sum of a list of numbers

listOfNumbers = [2,4,1,8,3,5]

total = 0 # initialise the summation variable

for num in listOfNumbers:
    total = total + num


print("The sum of", listOfNumbers,"is", total)
```

Exercise

- Implement a program that computes the sum over a list of numbers:

```
# compute the sum of a list of numbers

listOfNumbers = [2,4,1,8,3,5]

total = 0 # initialise the summation variable

for num in listOfNumbers:
    total = total + num
    print(total)

print("The sum of", listOfNumbers,"is", total)
```

Exercise

- Implement a program that computes the sum over a list of numbers:

```
# compute the sum of a list of numbers

listOfNumbers = [2,4,1,8,3,5]

total = 0 # initialise the summation variable

for num in listOfNumbers:
    total += num
    # total = total + num
    # print(total)

print("The sum of", listOfNumbers,"is", total)
```

Exercise

- Implement a program that computes the sum over a list of numbers:

```
# compute the sum of a list of numbers

#listOfNumbers = [2,4,1,8,3,5]
#listOfNumbers = [3,3,3,3,3,3]
#listOfNumbers = [3,5,2]
#listOfNumbers = []
listOfNumbers = [3.2,5.9,2.2]

total = 0 # initialise the summation variable

for num in listOfNumbers:
    total += num
    # print(total)

print("The sum of", listOfNumbers,"is", total)
```

Exercise

- Implement a program that tests whether all numbers in a given list are odd or not.

```
list_of_numbers = [2,4,8,6,3,2,4]

# ... your code ...

if all_numbers_are_even:
    print('all numbers are even')
else:
    print('some numbers are odd')
```

`break`, again

- It's sometimes convenient to know whether a loop was terminated by executing a `break` statement or not.
- Loops can have an optional `else` block at the end.
  - The `else` block is executed when the loop terminates <"properly" ...
  - i.e. unless the loop was terminated by executing a `break` statement.

```
for n in some_list_of_numbers:
    if n % 2 == 0:
        break
else:
    print("All numbers are odd")
...
```

`for` loops

```
string = str(input("Please input a string :"))

for ch in string:
    print(ch)
```