Python for Language Processing

Control Structures

Dr. Jakob Prange

Fakultät für Angewandte Informatik - Universität Augsburg

CL Fall School 24

Credit: This course is based on material developed by
Annemarie Friedrich, Stefan Thater, Michaela Regneri, and Marc Schulder at Saarland University

How to think about programming

- What is an algorithm?
- What is a program?
- Requirements for algorithms?

Imperative Programming Paradigm

- Imperative: *First do this, then do this.*
  Procedural Programming. Control Structures execute computational steps, state of the program changes as a function of time.
  Commands can be grouped into procedures.

Example

`Celsius_to_Fahrenheit(c)`

1. Multiply `c` with 1.8 and save result to `temp`.
2. Add 32 to `temp` and return result of this.

Elements of imperative programs

- Variables
- Assignments
- Expressions
- Control Structures: loops, branches

Values, Variables, Data Types

- Values may have different data types: numbers, lists, strings. . .

Variable Assignment

```
myList = [1, 2, 3, 4]
number = 4
text = 'hello'
number = 'world'
```

- Variables = **placeholders** for values.
- Variables point to positions in the memory where values are stored.
  Value of a variable can change over time. (Point to a different location or overwrite the memory location's value.)

Some Data Types

- Boolean: truth values: `True` and `False`
- Numbers: `int (2), float (2.0), complex`
- Strings: `str`
- Collections: `tuple, list, set, dict`

Imperative Programming Paradigm

- Imperative: *First do this, then do this.*
  Procedural Programming. Control Structures execute computational steps, state of the program changes as a function of time.
  Commands can be grouped into procedures.

Example

`Celsius_to_Fahrenheit(c)`

1. Multiply `c` with 1.8 and save result to `temp`.
2. Add 32 to `temp` and return result of this.

Elements of imperative programs

- Variables ✓
- Assignments ✓
- Expressions ✓
- Control Structures: loops, branches ⇐

Statements

- Python program = sequence of statements
- seen so far: assignments, `print` (actually, an expression)
- statement ≈ a step in the underlying algorithm
- separated by line breaks
- it is possible to write multiple statements in one line:
  then need to separate them by semicolons

Example

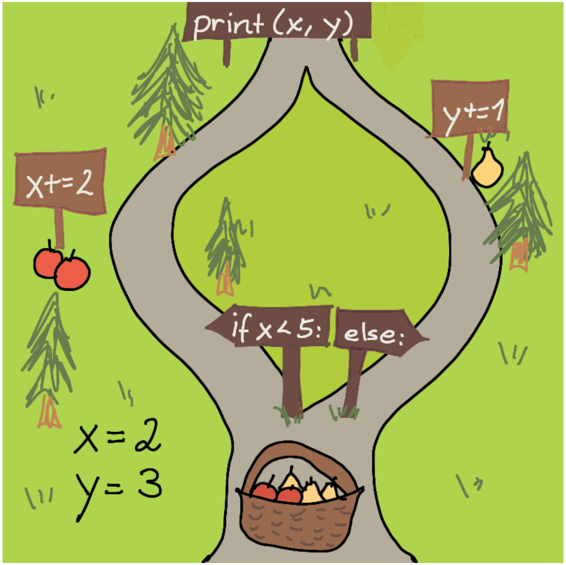
```
a = b = 2
a *= b; print(a)
b = a + b
```

Control Structures

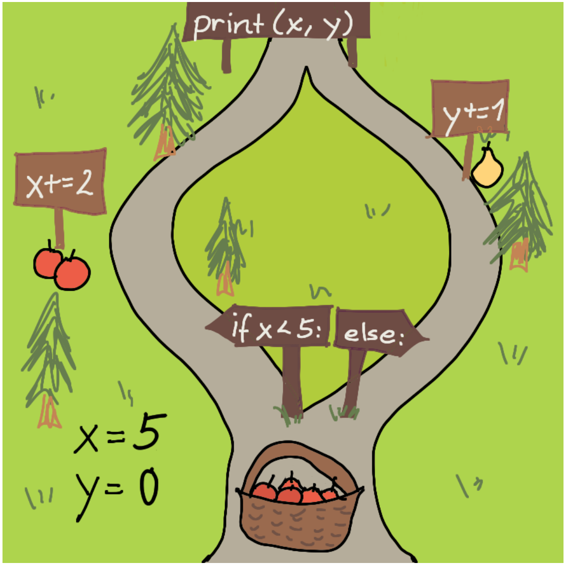
Sometimes we want to execute statements

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

Conditions: `if - else`



Conditions: `if – else`



Conditions: **`if – else – elif`**

```
if expr_1:
    block_1

if expr_1:
    block_1
else:
    block_2

if expr_1:
    block_1
elif expr_2:
    block_2
else:
    block_3
```

- if `expr_1` evaluates to `True`, `block_1` is executed
- Values evaluating to `False`: `False, 0`, the empty string (”), empty lists / sets...
- All other values are true.
- A block consists of one or more statements

Indentation

Spaces are important.

Indentation shows structure of code.

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

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

Blocks

- Block = grouping of statements
- instructions of the same block must be indented by the same number of the same type of whitespace characters (blank/tab)
- Best practice: always stick to the same type of whitespace!
  Using an IDE (e.g. PyCharm) makes your life easier.

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

Exercise 1: `if, else` and Blocks

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

```python
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 2: `if` VS `elif`

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

What does this script return for

(a) x = True; y = True

(b) x = False; y = True

(c) x = True; y = False

(d) x = False; y = False

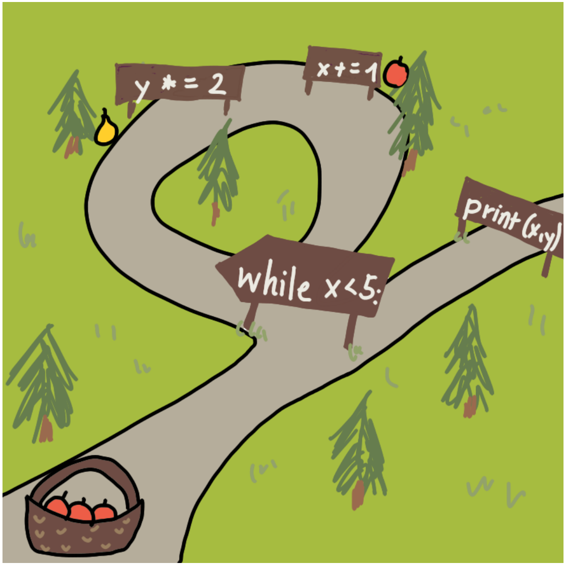
Exercise 3: `if` VS `elif`

```
1  if x:
2         print("Hello")
3  elif y:
4         print("World")
5  else:
6         print("Bye bye")
```

This script is slightly different from the last one (pay attention to line 3). Again, what does it do, given the following values:

- (a) x = True; y = True
- (b) x = False; y = True
- (c) x = True; y = False
- (d) x = False; y = False

Loops: **`while`**



Loops: **while**

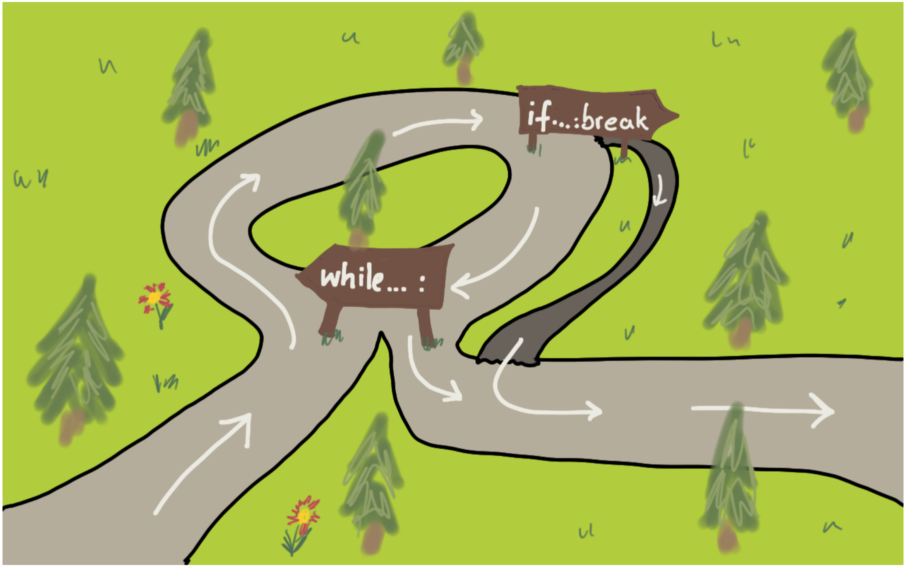
```
1  while expr:
2      block
```

- Evaluate `expr`.
- If `False`: continue program after loop (next statement with same indent as `while`)
- If `True`: execute statements of block. Then go back to line 1.

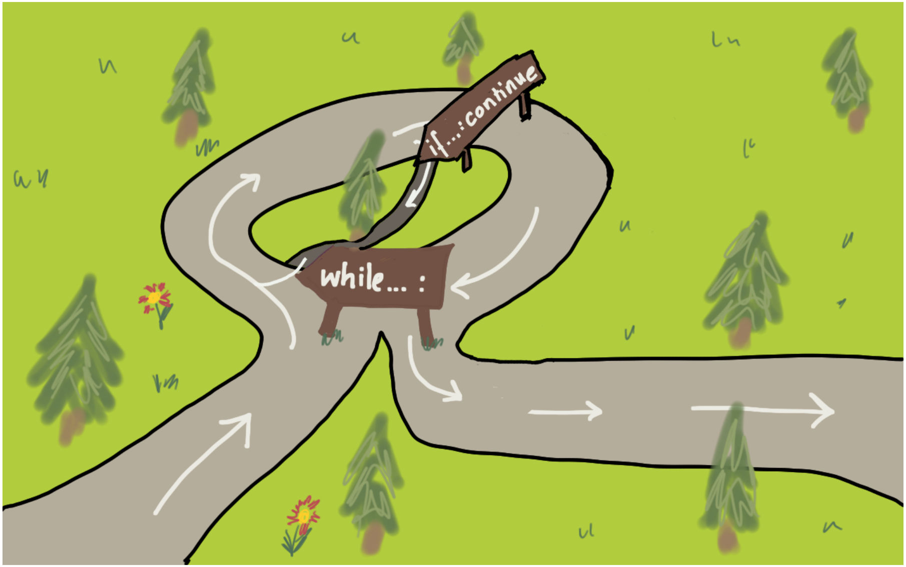
Exercise 4: What is the output of the following program?

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

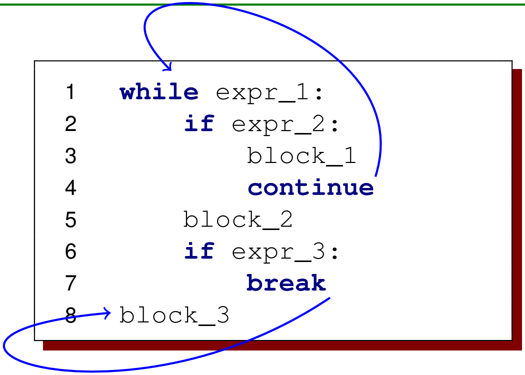
The `break` statement



The `continue` statement



Loops: `break` and `continue` Statements



```
while expr_1:
    if expr_2:
        block_1
        continue
    block_2
    if expr_3:
        break
block_3
```

- `break` exits the current loop without evaluating the condition
- `continue` skips the remainder of the current iteration, evaluates the condition again und continues the loop (if the condition is `True`)

Loops: `for`

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

- For now, you can imagine that `range(0,5)` creates a list: `[0, 1, 2, 3, 4]`
- Note: **`range(start, end)`**:
  The end point is not included in the sequence.
- **`range(start, end, step)`**: All arguments must be integers.
  `range(0,10,2)` returns `[0, 2, 4, 6, 8]`
  `range(10,0,-2)` returns `[10, 8, 6, 4, 2]`
- range() does not actually return lists, it returns an iterator (more about this later) - if you want get lists (e.g. for printing):

```python
x = list(range(0,2))
print(x)
```

Loops: `for`

```python
weekdays = ['Tuesday', 'Thursday']
for day in weekdays:
    print("Today is a", day)
```

- Python executes the block of the loop once per item of the list.
- The list item is assigned to the variable, here `day`.

Exercise 5: Loops

What is the output of the following program?

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