An introduction to Programming (in Python)

Kristoffer Nielbo

Center for Humanities Computing AArhus|chcaa.io
aarhus university, denmark

CENTER FOR HUMANITIES
COMPUTING AARHUS

Imperative

```python
class Person:
   def __init__(self, name, age=None, sex=None):
      self.name = name
      self.age = age
      self.sex = sex

   def says(self, message = '...'):
      print(f'{self.name}: {message}')

class Researcher(Person):
   def __init__(self, pay=10, areas=['research'],  **kwargs):
      super(Researcher, self).__init__(**kwargs)
      self.areas = areas

if __name__ == '__main__':
   kln = Researcher(
         name='Kristoffer L. Nielbo',
         age=44, sex='male',
         areas=['humanities computing', 'interactive HPC']
      )

   kln.says(message = 'hello and welcome to PftH21!')
```

Academic objectives

Knowledge:

- reflect on different programming techniques
- apply the key theoretical and methodological approaches of the course
- critically reflect on their own oral and/or written products (and those produced by others) in relation to the academic and theoretical discussions of the course.

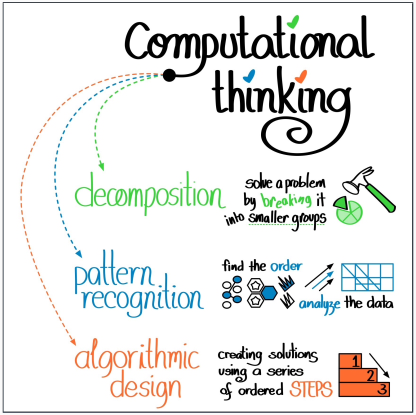
Skills

- apply Computational Thinking steps (i.e., problem decomposition, pattern recognition, abstraction, algorithm design) on problem-solving tasks
- discuss the theme of the course in the foreign language in an academic context.

Competences

- design and develop small computer programs using Python
- participate constructively in learning collaboration in a foreign language across educational and/or cultural backgrounds
- gain perspective on and compare the academic profile of the degree programme in relation to the subject areas of the humanities.

CT???



The ability to identify patterns, decompose large problems into small parts, develop algorithms to solve problems, and generalize to find solutions.

CT Example

Problem `12A4BA78AB11A1314AB`

Decomposition:

- Two types of data, numbers and letters
- Numbers (`int`) are in ascending order
- Letters only `A`, `B`, `AB`
- `AB` only occurs in the end

Pattern recognition:

- `A` in 3^{rd} positions
- `B` in 5^{th} positions
- Numbers are positional values (index)

CT Example

Validate:

x_i	1	2	A	4	B	A	7	8	A	B	11	A	13	14	AB
i	1	2	3	4	5	6	1	2	3	4	5	6	13	14	15

Design (possible) solution

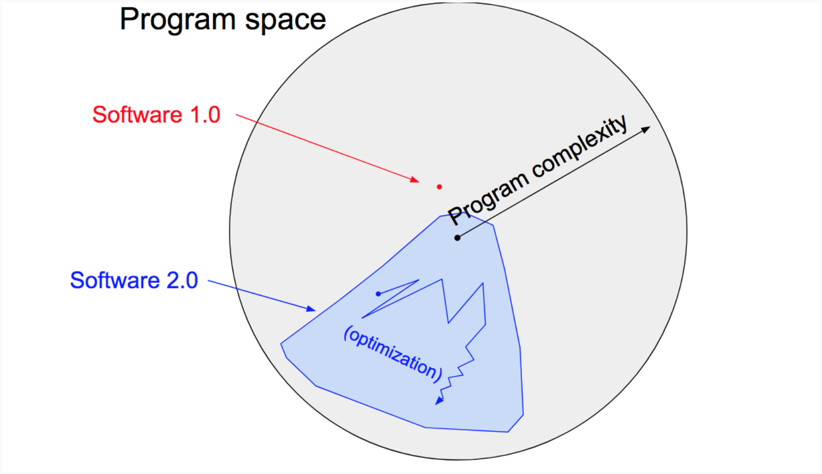
```python
def solution(n):
    result = ''
    for i in range(1, n + 1, 1):
        if i == n:
            result += 'AB'
        elif i % 3 == 0:
            result += 'A'
        elif i % 5 == 0:
            result += 'B'
        else:
            result += str(i)
    return result
```

Pannang Curry 1-2-3!

1. Open your right refrigerator door and remove ingredients from the following locations: Door shelf 2. Spot 1; Crisper drawer 1, Spot 3; Crisper drawer 1, Spot 5.
2. Open your third kitchen drawer and remove the utensils labeled ‘1’, ‘3’, ‘4’, ‘9’, and ‘12’.
3. Use your arms to apply utensil 1 to ingredients 1-3. Place ingredients inside utensil 3.

*Note: This recipe uses ShaKL the Shared Kitchen Layout. To use ShaKL, you’ll need to have installed ShaKL shelving, cabinetry, and utensils throughout your kitchen and pantry and have basic understanding of ShaKL managers.To learn more, read *Up and Running with ShaKL* (O’Billy Press, 2015). Want to improve ShaKL? Consider contributing to our team.

Program space



Inversion of input-output

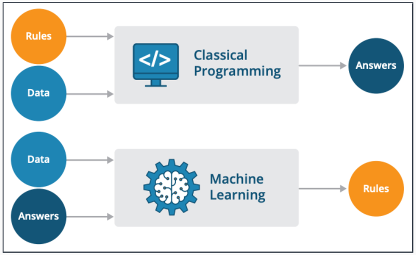


Figure: Software 1.0 involves manually writing rules. Software 2.0 is about learning these rules from data (credit: S. Charrington)

Andrej Karpathy
“they [neural networks] represent the beginning of a fundamental shift in how we write software.”

Programming Paradigms

Paradigm
A programming paradigm is a style or ‘way’ of programming that facilitates transfer of knowledge in code.

Imperative

```
sum = 0
for x in my_list:
   sum += x
print(sum)
```

Programming with an explicit sequence of commands that update state.

Declarative

```
SELECT SUM(my_column)
FROM my_database
```

Programming by specifying the result you want, not how to get it.

Procedural

```python
def do_add(any_list):
   sum = 0
   for x in any_list:
      sum += x
   return sum

print(do_add(my_list))
```

Imperative programming with procedure calls.

Object-oriented

```python
class ChangeList(object):
    def __init__(self, any_list):
        self.any_list = any_list

    def do_add(self):
        self.sum = sum(self.any_list)

create_sum = ChangeList(my_list)
create_sum.do_add()

print(create_sum.sum)
```

Object-oriented

Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces. Object orientation can be:

- Class-based: Objects get state and behavior based on membership in a class.
- Prototype-based: Objects get behavior from a prototype objects – alter class during run-time

```python
if questions:
    try:
        answer()
    except RunTimeError:
        pass
    else:
        print('break')
```