## Lecture I - Optimal Stopping

Programming: Everyday Decision-Making Algorithms

Dr. Tobias Vlćek and Dr. Nils Roemer

## **Optimal Stopping**

### **The Secretary Problem**

- · Imagine you're hiring a secretary
- · You must interview candidates one by one
- · After each interview, you must decide: hire or continue searching
- Once you reject a candidate, you cannot go back
- · How do you maximize your chances of selecting the best candidate?

#### **Real-World Applications**

- · Apartment hunting
- · Dating and relationships
- Job searching
- Parking spots
- · Selling a house

#### The 37% Rule



The optimal strategy is to: 1. Look at the first 37% of options 2. Remember the best one seen so far 3. Choose the next option that's better than the best seen

#### Why 37%?

```
import math

def optimal_stopping_point(n):
    return round(n/math.e, 2)

examples = [10, 20, 50, 100]

for n in examples:
    stop = optimal_stopping_point(n)
    percent = round(stop/n * 100, 1)
    print(f"For {n} options: Look at first {stop} ({percent}%)")
```

For 10 options: Look at first 3.68 (36.8%)

```
For 20 options: Look at first 7.36 (36.8\%)
For 50 options: Look at first 18.39 (36.8\%)
For 100 options: Look at first 36.79 (36.8\%)
```

## Implementation in Python

```
def optimal_secretary_strategy(candidates):
    n = len(candidates)
    lookout_phase = int(n/math.e)
    best_so_far = max(candidates[:lookout_phase])

for i in range(lookout_phase, n):
    if candidates[i] > best_so_far:
        return i
    return n-1 # If no better candidate found
```

## **Practical Examples**

### **Apartment Hunting**

Task: Let's simulate an apartment search in Hamburg

```
import random

def generate_apartments(n):
    """Generate n apartments with random scores (1-100)"""
    return [random.randint(1, 100) for _ in range(n)]

# Example usage
apartments = generate_apartments(10)
stopping_point = optimal_stopping_point(len(apartments))
print(f"Look at first {stopping_point} apartments before deciding")
```

Look at first 3.68 apartments before deciding

#### **Interactive Demo**

Let's try it!

```
def apartment_search_game():
    apartments = generate_apartments(10)
    for i, score in enumerate(apartments):
        print(f"\nApartment {i+1} Score: {score}")
        if i < int(len(apartments)/math.e):
            print("Still in observation phase...")
            continue

        choice = input("Take this apartment? (y/n): ")
        if choice.lower() == 'y':
            if score == max(apartments[i:]):
                 print("Great choice! This was the best remaining option!")
        else:
                 print("There was a better apartment later...")
            break</pre>
```

## **Variations**

## **Cost of Searching**

- · What if each additional search costs money?
- How does this change our strategy?
- Trade-off between finding better options and search costs

#### **Risk Tolerance**

- · Different scenarios require different strategies
- Conservative vs. aggressive approaches
- · Impact of wrong decisions

## **Questions?**

## **Further Reading**

- Christian, B., & Griffiths, T. (2016). Algorithms to live by: The computer science of human decisions
  Ferguson, T. S. (1989). Who solved the secretary problem?

## **About this Course**

#### About me

- · Field: Optimizing and simulating complex systems
- · Languages: of choice: Julia, Python and Rust
- · Interest: Modelling, Simulations, Machine Learning
- Teaching: OR, Algorithms, and Programming
- · Contact: vlcek@beyondsimulations.com



Tip

I really appreciate active participation and interaction!

#### **Course Outline**

- Part I: Introduction to Programming with Python
- · Part II: Data Science Tools with Python
- · Part III: Programming Projects

### **Participation**

- · Prequisite for course Management Science (Prof. Goel)
- Try actively participating in this course
- You will find it much (!) easier to follow Prof. Goel
- · Materials will be provided in the KLU portal
- Slides are hosted at beyondsimulations.github.io/Introduction-to-Python

### **Teaching**

- · Lecture: Presentation of tools and concepts, based on small examples and code snippets
- · Tutorial: Hands-on examples to be solved in groups
- · Difficulty: Difficult at first, but gradually easier

### **Passing the Course**

· Pass/fail course

- 75% attendance required for passing the course
- 2 assignments and 1 little project
- · You will be given programming exercises to solve
- · You can group up (3 students) and work together
- · Each student group submits one solution

#### **Solution**

- Provide a code solution to the problem (.py files)
- · Code files need to be executable
- · Detailed explanations of your code should be provided
- Use comments or docstrings in your code
- · Provide a general (verbal) introduction to each problem

. . .



I'd encourage you to start and submit your solution early

#### **Difficulty of the Course**

- We'll cover the basics of programming (in Python) at first
- This is similar to learning a new foreign language
- · First, you have to get used to the language and learn words
- Later, you'll be able to apply it and see results
- Similar to learning a language: Practice, practice, practice!

### What to expect

- Some investment in the beginning to see the return later
- You can ask questions and get support anytime
- · After completion, you will be able to read code
- · and write your own program using Python
- · That's quite something!

#### **Goals of the Course**

- · Essential concepts and tools of modern programming
- · Automated solutions for recurrent tasks
- Algorithm-based solutions of complex problems
- · Usage of AI in a specific context

### **Python as Language**

- Origins: Conceived in late 1980s as a teaching and scripting language
- Simple Syntax: Python's syntax is straightforward and easy to learn
- Versatility: Used in web development, data analysis, artificial intelligence, and more
- · Community Support: A large community of users and extensive documentation

## **Help from Al**

- You are allowed to use AI (GitHub Copilot, ChatGPT, LLama3 ...)
- These new tools are really powerful for learning Python!
- · They can help you a lot to get started with programming



Warning

But you should not simply use them to replace your learning.

# Why learn programming?

## **Analytics**

Photo by Choong Deng Xiang on Unsplash

#### Research

Photo by National Cancer Institute on Unsplash

#### **Visualization**

Photo by Clay Banks on Unsplash

#### **Finance**

Photo by Ishant Mishra on Unsplash

## Logistics

Photo by Denys Nevozhai on Unsplash

## How to learn programming

#### My Recommendation

- 1. Be present: Attend the lecture and participate
- 2. Put in some work: Repeat lecture notes and try to understand the examples yourself
- 3. Do coding: Run code examples on your own, play around, *google*, modify, and solve problems on your own

. . .



Great resources to start are books and small challenges. You can find a list of book recommendations at the end of the lecture. Small challenges to solve can for example be found on Codewars.

#### Don't give up!

- Programming is problem solving, don't get frustrated!
- · Expect to stretch your comfort zone

### **Learning Path**

- The learning path can be quite steep!
- · First of all help each other!
- Try to find help in lecture materials and books, the Python documentation, and online (e.g. Google, ChatGPT, StackOverflow, ...)
- In case you get frustrated, read the following helpful blog post about the challenges on medium.com

#### **Errors**

In case you find errors and typos in the lecture notes, please report them in the following form: https://tally.so/r/w7oapa

## **Setting up Python**

#### **Install Python**

- · Sources are the Python website or Anaconda
- · But I would recommend we start by installing Thonny
- · An open source IDE running on Windows, Linux and Mac
- · Built in Python interpreter and package management!

#### What is an IDE?

- Integrated Development Environment = application
- · It allows you to write, run and debug code scripts
- Thonny is an IDE specifically for Python beginners
- · It is easy to use as beginner!
- Other IDEs include for example PyCharm from JetBrains or Visual Studio Code from Microsoft

#### Python on iPads

- · Although you can run Python scripts from on your iPad, it is not recommended for the course
- · Nonetheless, you could use Pythonista
- · It works locally on your iPad and can run most scripts

#### Caution

Not all packages available in Python are available in Pythonista, thus you might need a computer to solve certain problems.

## Your first code

#### Hello, World!

Task: Create a directory for the course and create a new file called hello\_world.py with the following code:

```
# This is a comment in Python
print("Hello, World!")
```

```
Hello, World!
```

. . .

Run it with the green 'run' button or by pressing F5!

. . .

#### Note

"Hello world" is a classic example to start with. It is often used as a test to check if your computer is working properly and that you have installed the necessary software.

### Hello, World in a Message

Task: Change the code in your hello\_world.py file. Assign the string "Hello, World!" to a variable called message and print the variable.

. . .

- Use the equals sign (=)
- · Variable name goes on the left
- · Value to be assigned goes on the right

. . .

```
# Here we assign the string "Hello, World!" to variable message and print it
message = "Hello, World!"
print(message)
```

Hello, World!

#### **Hello, World in Parentheses**

We can also mix " and ' in a string, if we are consistent:

```
# This code works
message = 'I shout "Hello, World!"'
print(message)

I shout "Hello, World!"
...

# This code does not!
message = 'I shout 'Hello, World!""
print(message)
```

Try it yourself! What does happen, if you try to run it?

#### **First Errors**

SyntaxError: invalid syntax

- · The code is not valid Python syntax
- · Likely the most common error that you will encounter!
- · Happens when you make a mistake, e.g., illegal character, missing a colon, parentheses or quotations
- You can fix this by correcting the code and re-running
- · In the lecture you will encounter many more errors!

## **Program**

#### What is a Program?

- Sequence of instructions telling a computer what to do
- · Written in a language the computer can understand
- · Basic operations in most languages:
  - Input: Data from keyboard, file, network, sensors, etc.
  - Output: Display data, save it, send over network, etc.
  - **Processing**: Perform calculations, analyze data, make decisions, find patterns, etc.

### **Key concepts**

- Key concepts in most languages:
  - Variables: Store and manipulate data
  - Conditional execution: Check and execute accordingly
  - Loops: Perform actions repeatedly
  - Functions: Group instructions for reusability

. . .

Note

We will cover these concepts in more detail later in the course.

### **How Python executes code**

- Python is an interpreted language
- · The source code is executed line by line
- The interpreter checks the syntax and executes the code
- This is in contrast to compiled languages, where the code is compiled into machine code before execution

### Hello again, World!

Let's go back to our first program:

```
# Our first program
message = "Hello, World!"
print(message)
```

- Comment: In the first line we define a comment with #
- Variable: In the second we define a variable message
- Function: In the third line we call a function print

## Don't worry!

- Already confused? Don't worry about it for now!We'll learn more about variables and functions later

## **Python's Syntax**

### The Zen of Python

- Python's name originally comes from Monty Python
- · Style is based on a philosophy called Zen of Python
- · Collection of 19 statements with general principles

. . .

Task: Try this code in Python:

```
# Try this code in Python to see the Zen of Python import this
```

#### **Variables**

- · A variable in Python is a name that points to a value
- Created by using the assignment operator =
- Python does not require a declaration of variable types

```
a = 2 # Variable a assigned the value 2
b = "Time" # Variable b assigned the value "Time"
c = print # Variable c assigned the print function
c(b) # Now we can call the print function with c
```

Time

. . .

But there are certain rules to variable names!

### **Variable Naming Conventions**

- Must start with a letter or underscore \_
- · Can contain letters, numbers and underscores
- · Names are case sensitive, e.g., a and A are different!
- Cannot be a reserved word, e.g., for, if, def, etc
- · Good names are short and meaningful for humans!

. . .

Question: Which of the following fulfill these conditions? a, \_duration, 1x, time\_left, 1\_minute, oneWorld, xy4792

#### **Functions**

- · Functions are named blocks of code
- Can take arguments function([arguments])
- Can return results or None

. . .

```
# Print is such a function
print("Hello, World!") # It takes an argument and prints it to the console
print("Hello", "World!", sep=", ") # It can also take multiple arguments
```

```
Hello, World!
Hello, World!
```

. . .

#### Note

We will cover functions in more detail later in the course.

## Values and Types

#### What are Values and Types?

- · Value: Fundamental thing that a program manipulates
  - In Python, values are either numbers or strings
- Type: Type of a value
  - Determines what operations can be performed on it
  - type() is a function that returns the type of a value
  - Takes one argument and returns its type as string

#### **Strings**

Back to our example of "Hello, World!"

```
# We define the variable message and assign it the value "Hello, World!"
message = "Hello, World!"

# We save its type in another variable called message_type
message_type = type(message)

# We print the value of our new variable
print(f"{message} is a {message_type}")
```

Hello, World! is a <class 'str'>

. . .

Result: "Hello, World" is a string - in short 'str'.

. . .

But what about the f"?

## **Formated Strings**

- f-strings are strings that start with f
- They contain expressions, e.g., variables, in braces
- · Evaluated at run time and inserted into the string

. .

#### Note

In older code bases, f strings were not available. Here, interpolation could be done as shown below with print() and .format(). But this method is less concise and arguably less readable.

print("{} is a {}".format(message, message\_type))
Hello, World! is a <class 'str'>

#### **Specifying Formatted Strings**

- We can further specify their forman with {<to\_print>:<width>.<precision>f}
- · width can be a number specifying the output width
- <, ^, > can be used before the width to align the text
- precision can be used to specify the decimals
- · .f can be used to format floats

```
x = "hello"
print(f"{x:<10} has {len(x):>10.2f} characters.")
hello has 5.00 characters.
```

#### **Expressions**

- · Produce a value when evaluated
- · Can be used as part of larger expressions or statements
- · Statements are expressions that don't produce a value
- · Examples: arithmetic operations, function calls, variables

```
x = 1 # Statement that assigns the value 3 to x y = x + 2 # Expression on the right side assigned to a variable y print(f"Great, the result is \{y\}")
```

Great, the result is 3

#### A step back: What is a String?

- Remember: "Hello, World" is a string in short 'str'
- A string is a sequence of characters enclosed in quotes
- Examples: "Hello", 'World', "123", '1World23'

```
hello = "Hello"
world = 'World!'
print(hello,world,sep=", ") # We can specify the separator with the argument sep
Hello, World!
```

#### Note

Strings are immutable, we can't change characters in them once created.

#### **String Operations**

- · But we can also do much more with strings!
- String concatenation, indexing, slicing, length, etc.

```
two_strings = "Hello" + ", " + "World!" # String concatenation
print(two_strings)

Hello, World!
...
print(two_strings[0]) # Indexing starts at zero!

H
...
print(len(two_strings)) # With len we can find the length of our string

13
...
print("--x--"*3) # We can also repeat strings
```

#### **Booleans**

- Booleans represent two values: True and False
- · Internally they are represented as 1 and 0, respectively
- They are used for logical operations and control flow
- E.g.: if, while, for, elif, 'else

```
x = True
y = False
print(x)
print(type(y))
```

```
<class 'bool'>
...
```

> More on them in our next lecture!

## **Integers and Floats**

• Integers are whole numbers, e.g.: 1, -3, 0 or 100

- Floats are decimal numbers, e.g.: -4.78, 0.1 or 1.23e2
- Bit size is **not specified** (e.g.: 64 bits) in Python!

x = 1; y = 1.2864e2 # We can separate multiple operations in one line with semicolons print(f"{x} is of type {type(x)}, {y} is of type {type(y)}")

1 is of type <class 'int'>, 128.64 is of type <class 'float'>

#### Warning

The interpreter will automatically convert booleans to integers to floats when necessary, but not the other way around!

## **First Functions and Operators**

#### **Arithmetic operators**

Result: addition is 3
Result: substraction is -1
Result: multiplication is 12
Result: division is 1.75
Result: floor\_division is 1
Result: exponentiation is 3.0
Result: modulo is 1

#### **Precedence**

- Operators are the same as in most other languages
- · Can be combined with each other and variables
- Normal rules of precedence apply

. .

```
# Operator precedence works as on paper
combined_operation = 2 + 3 * 4
print(f"2 + 3 * 4 = {combined_operation}")

2 + 3 * 4 = 14
...
# Parentheses change precedence as expected
parentheses_operation = (2 + 3) * 4
print(f"(2 + 3) * 4 = {parentheses_operation}")
```

```
(2 + 3) * 4 = 20
```

### The input() Function

- · Used to get user input as string from the console
- Syntax: input([userprompt])
- · Displays optional prompt and waits for user input

. .

```
name = input("What's your name? ")
print(f"Hello, {name}!")
```

. . .

Important

The function always returns the input as string!

#### **Input in Action**

Task: Solve the following task:

```
# TODO: Ask the user for their age and print a compliment
...
Solution
age = int(input("How old are you? "))
print(f"You look great for {age}!")
```

### **Type Conversion**

Use type conversion for other data types

```
1. Integer: int(input())
2. Float: float(input())
3. Boolean: bool(input())
4. String: str(input())

...

# Converting to Integer
age = int(input("Enter your age: "))
next_year = age + 1
print(f"Next year, you'll be {next_year}")
```

#### The round() Function

```
# Hence, we can use the int() function to convert a float into an int
soon_int = 1.789
print(f"{soon_int} converted to {int(soon_int)} of type {type(int(soon_int))}")
```

. . .

```
# We can also use `round()` to round a float to an int
soon_int = 1.789
print(f"{soon_int} converted to {round(soon_int)} of type {type(round(soon_int))}")

1.789 converted to 2 of type <class 'int'>
...

# Or to a float with a certain number of decimals
no_int = 1.789
print(f"{no_int} converted to {round(no_int,1)} of type {type(round(no_int,1))}")

1.789 converted to 1.8 of type <class 'float'>
```

#### **Questions?**

. . .

Note

#### That's it for todays lecture!

We now have covered the basics on the Python syntax, variables, and data types.

## Literature

#### **Interesting Books to start**

- Downey, A. B. (2024). Think Python: How to think like a computer scientist (Third edition). O'Reilly. Link to free online version
- Elter, S. (2021). Schrödinger programmiert Python: Das etwas andere Fachbuch (1. Auflage). Rheinwerk Verlag.

. . .

#### **i** Note

Think Python is a great book to start with. It's available online for free here. Schrödinger Programmiert Python is a great alternative for German students, as it is a very playful introduction to programming with lots of examples.

. . .

For more interesting literature to learn more about Python, take a look at the literature list of this course.