Lecture 1

Introduction, Advanced programming

Ph. D, cand polyt Bjarne Poulsen

https://orbit.dtu.dk/en/persons/bjarne-poulsen(79310464-faeb-4346-baa2-a444be17d1f3).html

Advanced programming

- Bjarne Poulsen

Introduction to

What this course is all about!

Introduction to Advanced programming

- Course objectives
- Learning objectives
- project
- Course literature
- Python
- Course overview

Advanced programming

- Course objectives
  - An introduction to Python and mazes.
  - Provide a basic knowledge of Python and mazes order to understand the design and working of most IT.

- Learning objectives
  - Python and mazes systems and fundamental concepts.
  - Make a project.

Advanced programming

- project
  - Elaboration of a project with a maximum of 6 standard pages to clarify concepts.
  - Evaluating the Efficiency of Maze-Solving Algorithms using Python.

”How can it be determined which pathfinding algorithm is the most efficient, based on adjustable criteria?

We will learn about

- Python’s basic data types, basic data structures, control structures, expressions, statements, operators, and program operation……
- - algorithm templates
- - design patterns
- - collections
- - development environment
- - test
- - architecture
- - advanced data structures and program libraries
- - functional programming paradigm and integration with OOP
- - parallel programming.

Maze

- The algorithms that are to be used for the purpose of generating mazes in the application, are:
  - Eller's algorithm
  - Randomized Prim's algorithm
  - Depth-First generation algorithm (DFG algorithm)

- For the purpose of solving mazes, the following algorithms are:
  - Dijkstra's algorithm
  - A-Star algorithm (A* algorithm)
  - Depth-First solving algorithm (DFS algorithm)

Algorithm Requirements:

1. Generate maze of a given size
2. Solve maze of a given size
3. Data for the algorithmic decision-making used by the solving algorithm should be saved
4. The found solution for each maze should be saved
5. The time complexity of the solving algorithms should be saved
6. The criteria should be combined into an overall score for the algorithms

Based on these requirements, the workflow of the algorithms in the application should be as follows in figures

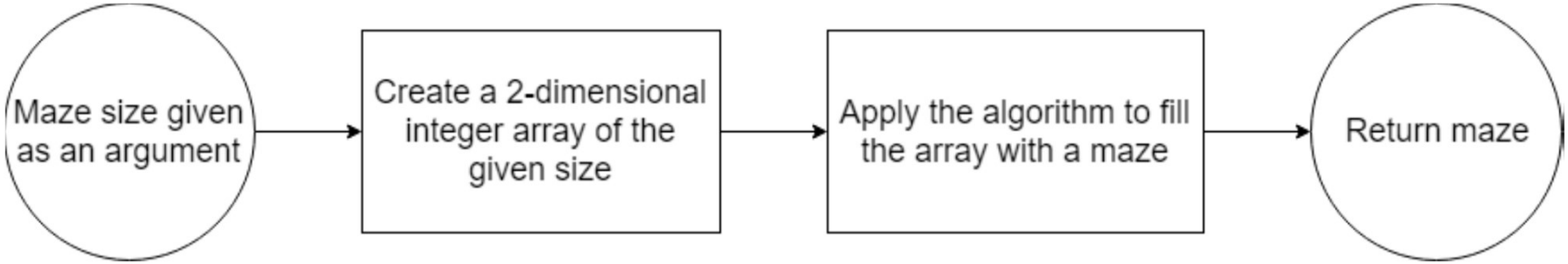


Diagram of the initial workflow design for maze generation, based on the requirement specifications that have been proposed.

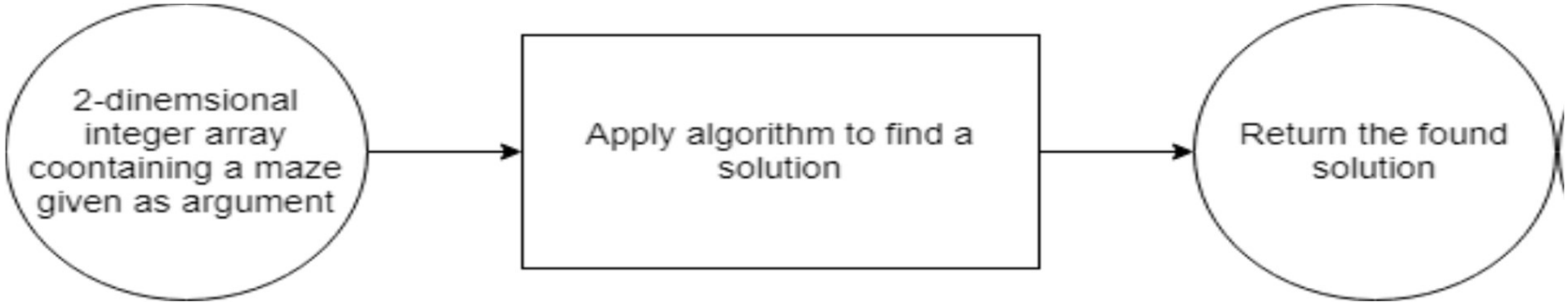


Diagram of the initial workflow design for maze solving, based on the requirement specifications that have been proposed.

Why Python?

There are many good and powerful programming languages in the world. Usually, each language is good for a certain type of job.

Python's syntax is quite straight forward, allowing you to write clean code, which is easily understood by others and which can be easily used, extended and debugged.

Python is used in many domains, for example to
creating games
build web applications
solve business and scientific problems

It is used in incredibly many projects, see

https://www.python.org/about/success/
http://pythonhackers.com/open-source/
https://docs.python.org/3/faq/general.html#what-is-python

See some of the following systems, which are build in Python.

Tensorflow https://github.com/Tensorflow/Tensorflow
Django https://github.com/Django/Django
Ansible https://github.com/Ansible/Ansible
Reddit https://github.com/reddit/reddit
Celery https://github.com/celery/celery
Kivy https://github.com/kivy/kivy

It has a great, welcoming, and helpful community. See for example:

http://stackoverflow.com/questions/tagged/python
https://www.reddit.com/r/learnpython/

You are in good society ;) *"Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities"*

http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext

Python is a great language to learn!

What is Python?

Python is an

- Interpreted.
- Interactive.
- object-oriented programming language.

It incorporates **modules**, **exceptions**, **dynamic typing** (varible bound only to an object - not to a data type), and classes.

Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface.
Finally, Python is **portable**: it runs on many Unix variants, on the Mac, and on Windows 2000 and later.

The name Python comes from the surreal British comedy group Monty Python, not from the snake

Literature

You should not really need any other literature besides all the information given in the course and what you can find online.

Python Crash Course A Hands-On, Project-Based Introduction to Programming by Eric Matthes
(https://www.nostarch.com/pythoncrashcourse)

Automate the Boring Stuff with Python Practical Programming for Total Beginners by Al Sweigart
(https://www.nostarch.com/automatestuff)

A Whirlwind Tour of Python
(http://www.oreilly.com/programming/free/a-whirlwind-tour-of-python.csp)

The Exercises

Today, we will form groups of 3 students.

Tasks for student groups

You will get assignments created by the teachers that relate to your learning objectives.

You will then:

- **Create** a review group.
- **All groups implement a solution** for each given problem.
- **Each group reviews** the another group solution and documenting.
- **Each group presents their solution**.

The Exam

- Group presentation of 10 minutes followed by 10 minutes of questions per student (minimum 30 minutes).

- The exam is based on a presentation of the student's group project and it is facilitated by an interactive programming environment. Additionally, this includes a discussion of the project's solutions with respect to the main topics of the learning objectives.

- The questions will be learning objectives of the course.

What do I need for this course?

You will need:

A Python installation via the Anaconda distribution.
The editor Visual Studio Code.

A Python Installation

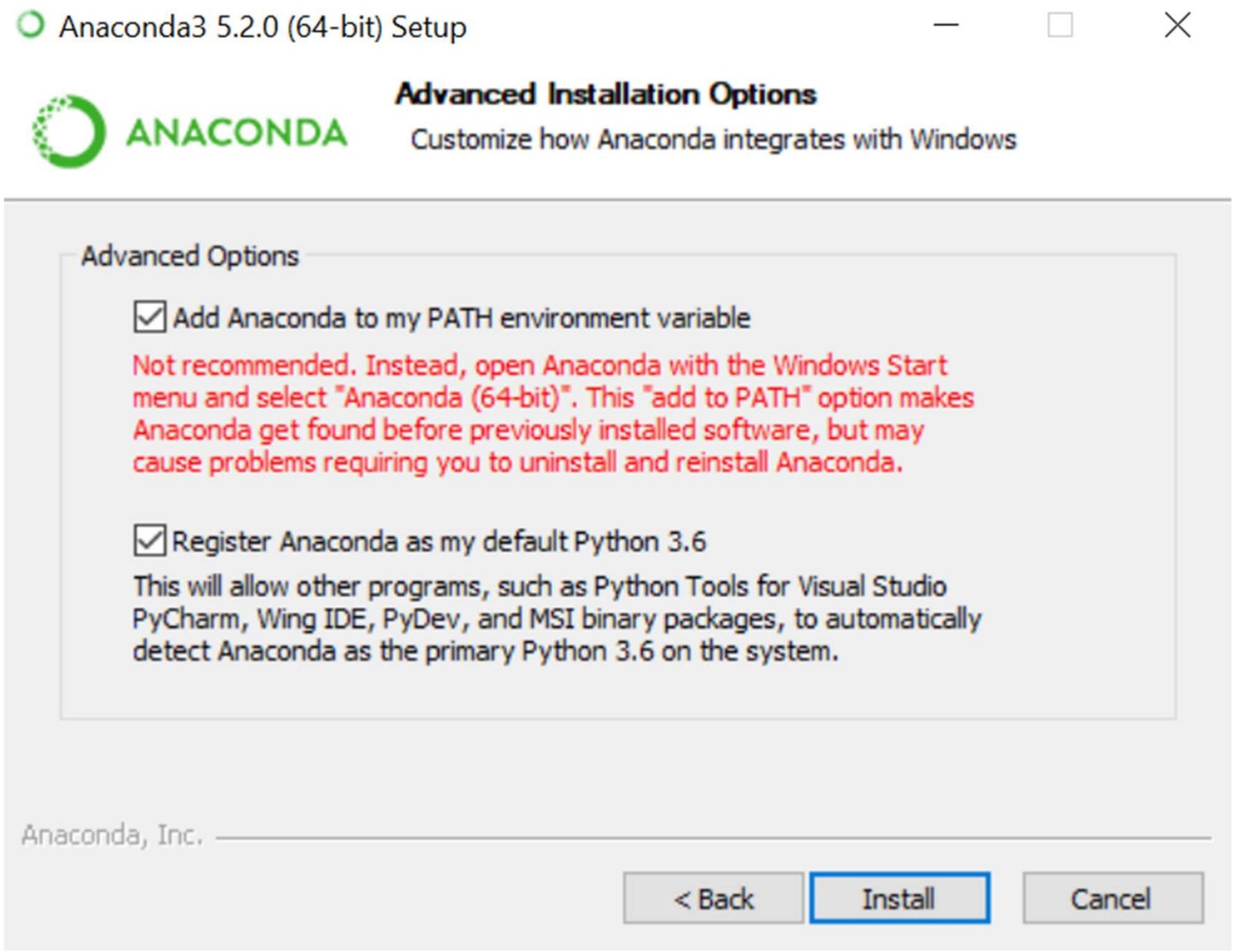
Likely, the easiest way of installing and configuring Python is via the distribution *Anaconda*.

Install the Python 3.7 version corresponding to your machine and OS from
https://www.anaconda.com/download.
In case you need help read the installation instructions here:
https://docs.anaconda.com/anaconda/install/

OBS: This course will make use of Python 3.X. Many of the provided code examples will not work with Python 2!

Installing Anaconda on Windows.

When starting the installation choose the settings as shown below.


Anaconda3 5.2.0 (64-bit) Setup
ANACONDA
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda to my PATH environment variable
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda as my default Python 3.6
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.6 on the system.
Anaconda, Inc.
< Back
Install
Cancel

Testing your Python Installation

On the terminal/GitBash try to check the version of Python, which should be something similar to:

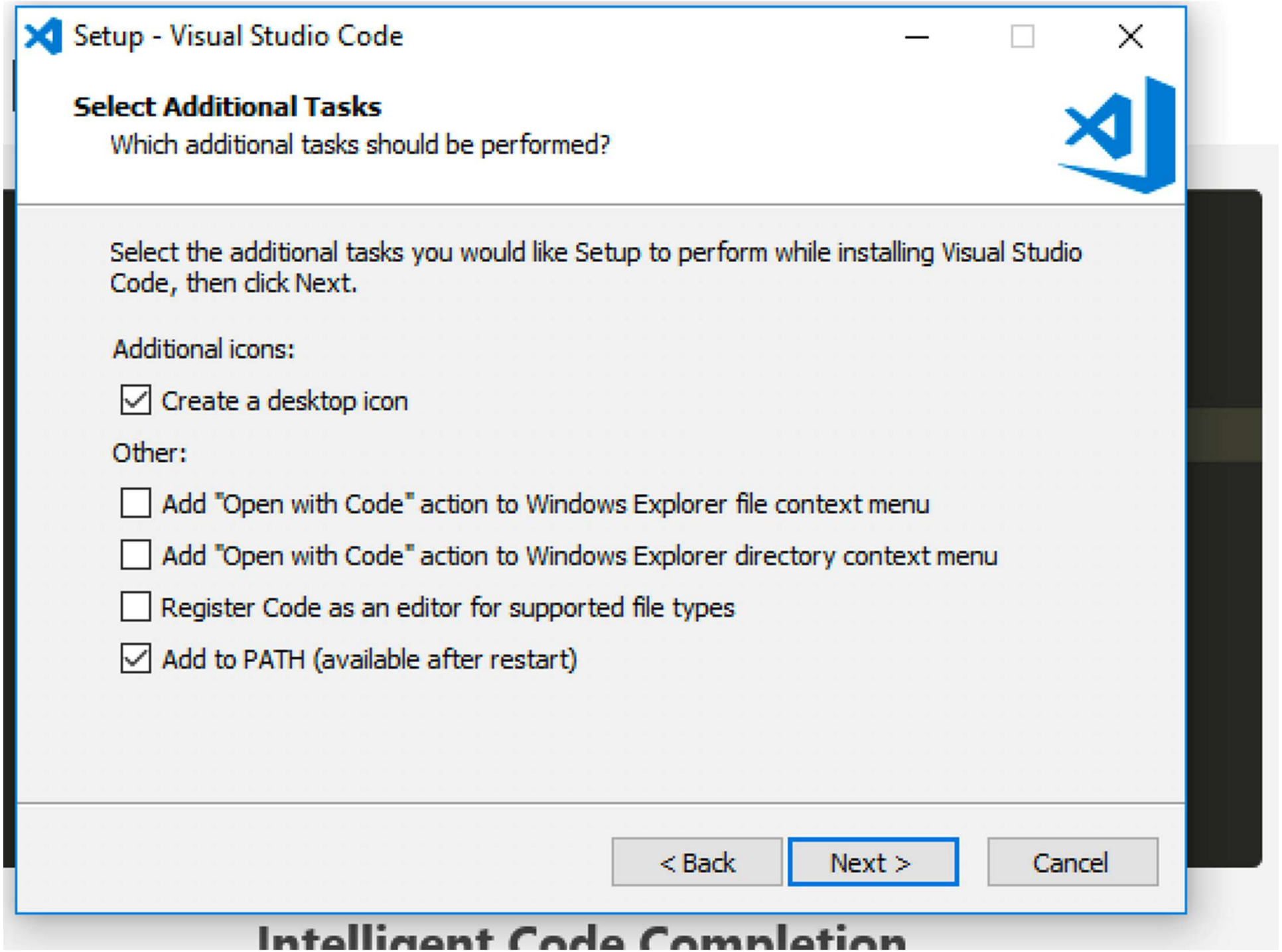
```
$ python --version
Python 3.6.4 :: Anaconda, Inc.
```

Visual Studio Code

Installation on Windows

In case you are working from home and do not have access to one of the USB keys, navigate to https://code.visualstudio.com/ and download the installer by clicking the big green download button.

The installation is straight forward, the only thing that is important is to check "Add to Path" as in the image below.


Setup - Visual Studio Code
Select Additional Tasks
Which additional tasks should be performed?
Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.
Additional icons:
Create a desktop icon
Other:
Add "Open with Code" action to Windows Explorer file context menu
Add "Open with Code" action to Windows Explorer directory context menu
Register Code as an editor for supported file types
Add to PATH (available after restart)
< Back
Next >
Cancel
Intelligent Code Completion

Running Python Code

Basically, you have two posiblities to run Python code. Either you execute statements iteratively in the read-eval-print-loop (REPL) or you run self-contained programs.

The Python Interpreter

On the VM, you can run Python directly on the command line via the python command.

```
$ python
Python 3.6.3 |Anaconda custom (64-bit)| (default, Oct  6 2017, 12:04:38)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello world!')
Hello world!
```

This brings you to the Python REPL and lets you execute your first statements and evaluate your first expressions

IPython

Since the Python REPL might feel a bit "nacked" and bare metal you might want to use IPython (http://ipython.org), a bit more interactive coding environment. It provides you for example code completion via the tab key or direct help to functions. More to the use of IPython later.

```
$ ipython
Python 3.6.3 |Anaconda custom (64-bit)| (default, Oct  6 2017,
12:04:38)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('Hello world!')
Hello world!
In [ ]:
str.
```

Running Self-contained Python Programs

To create a simple "Hello World" script copy and paste the following code to your command line:

```
$ cat <<EOF > hello.py
print('Hello World!')
EOF
```

That is, with subsequently runnning the program, you should see something like in the following.

```
$ cat hello.py
print('Hello World!')
$ python hello.py
Hello World!
```

BREAK

A Whirlwind Tour through Python!

Marks Comments
Your code is stored in plain text files, which usually end on .py.

```
In [ ]:  # I am a comment
```

```
In [ ]: print('Hello World!')  # prints a string to stdout
```
End-of-Line Terminates a Statement

```
In [ ]: print('Hello World!’)
        print('Next Statement')
```
Alternatively Semicolon Terminates a Statement
However, do not do this! It is considered a bad practice.

```
In [ ]: print('Hello World!'); print('Next Statement')
```

Expressions with Basic Operators

Math operators from highest to lowest precedence:

Operator	Operation	Example	Evaluates to...
**	Exponent	2 ** 3	8
%	Modulus	22 % 8	6
//	Integer division	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 3	5

```
In [ ]: 2 ** 8
In [ ]: 23 // 7
In [ ]: 23 % 7
In [ ]:(5 - 1) * ((7 + 1) / (3 - 1))
```

Integer, Floating-point, and String Data Types

Common data types

Data Type	Examples
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.25, -0.5, 0.0, 0.5, 1.0
Strings	'a', 'Hello!', '11 things'

```
In [ ]: type(1)
In [ ]: type(1.0)
In [ ]: type('Hello!')
```

Strings

A string is simply a series of characters. Anything inside quotes is considered a string in Python, and you can use single or double quotes around your strings.

```
In [ ]: 'Hello world!'
In [ ]:"Hello world!"
In [ ]:"Hello world!" == 'Hello world!'
```

Mixing string delimiters

```
In [ ]: '"And God created great whales." —Genesis. —Moby Dick.'
In [ ]:"'And God created great whales.' —Genesis. —Moby Dick."
```

String Concatenation and Replication

```
In [ ]: 'Hello' + 42
In [ ]: 'Hello' * 4
```

Other Operations on Strings

```
In [ ]: name = ”ada lovelace”
        print(name.title())
In [ ]: len(name)
In [ ]: name = ”Ada Lovelace”
        print(name.upper())
         print(name.lower())
In [ ]: print(”1. Ada Lovelace”)
        print(”2. \tAda Lovelace”)
In [ ]: name = ”     Ada Lovelace ”
        name.lstrip()
In [ ]: name = ”Ada Lovelace      ”
        name.rstrip()
In [ ]: name = ”     Ada Lovelace     ”
        name.strip()
```

The None Value

In Python there is a value called None, which represents the absence of a value. None is the only value of the NoneType data type. Just like the Boolean True and False values, None must be typed with a capital N. This value-without-a-value can be helpful when you need to store something that won't be confused for a real value in a variable.

In [ ]: None == 5

Naming and Using Variables

When you're using variables in Python, you need to adhere to a few rules and guidelines. Breaking some of these rules will cause errors; other guidelines just help you write code that's easier to read and understand. Be sure to keep the following variable rules in mind:

Variable names can:

- contain only letters, numbers, and underscores
- start with a letter or an underscore
- not start with a number
- not contain spaces

Guidelines:

Avoid using Python keywords and function names as variable names!

Variable names should be short but descriptive.

for Loops Iterating over Lists and Sequences

```
for i in [0, 1, 2, 3]:
    print(i)
0
1
2
3

for i in range(4):
    print(i)
0
1
2
3
```

Iterating over Lists and Sequences with Indices

In case you want to iterate over a list or sequence of values in and you need to have access to the index of each element, you could do the following:

```
for idx, value in enumerate(range(4, 0, -1)):
    print(idx, value)
0 4
1 3
2 2
3 1
```

Enumerate() in Python

Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

Syntax:

enumerate(iterable, start=0)

Parameters:
Iterable: any object that supports iteration
Start: the index value from which the counter is
to be started, by default it is 0

Checking Lists for Element Containment

```
1 in [0, 1, 2, 3]
True

1 in range(4, 0, -1)
True

'Hej' in [0, 1, 2, 3]
False

1 in ['Call', 'me', 'Ishmael']
False

'Hej' in ['Call', 'me', 'Ishmael']
False
```

while Loops

The for loop takes a collection of items and executes a block of code once for each item in the collection

It has the syntax:

```
# counting to five
current_number = 0
while current_number <= 5:
        print(current_number)
        current_number += 1
```

0
1
2
3
4
5

Exercise

Write a program that creates gramatically valid English sentences.
Here, we consider a sentence to be gramatically correct when it follows the simple English grammar of the form:

Article Adjective Noun Verb.

That is, even the sentence A insect fly. is, for the moment, considered correct.
Create additionally to the exercise above, a list of definite and indefinite articles and verbs from an online resource.
Extend the above program to generate all possible sentences with the given words.

```
import random
# https://www.espressoenglish.net/100-common-adjectives-in-english/
adj = ["other", "new", "good", "high", "old", "great", "big", "American",
      "small", "large", "national", "young", "different", "black", "long",
      "little", "important", "political", "bad", "white", "real", "best",
      "right", "social", "only", "public", "sure", "low", "early", "able",
      "human", "local", "late", "hard", "major", "better", "economic",
      "strong", "possible", "whole", "free", "military", "true", "federal",
      "international", "full", "special", "easy", "clear", "recent",
      "certain", "personal", "open", "red", "difficult", "available",
      "likely", "short", "single", "medical", "current", "wrong", "private",
      "past", "foreign", "fine", "common", "poor", "natural", "significant",
      "similar", "hot", "dead", "central", "happy", "serious", "ready",
      "simple", "left", "physical", "general", "environmental", "financial",
      "blue", "democratic", "dark", "various", "entire", "close", "legal",
      "religious", "cold", "final", "main", "green", "nice", "huge",
      "popular", "traditional", "cultural"]

articles = ["the", "a", "an"]
```

```
# https://www.espressoenglish.net/100-common-nouns-in-english/
nouns = ["time", "year", "people", "way", "day", "man", "thing", "woman",
        "life", "child", "world", "school", "state", "family", "student",
        "group", "country", "problem", "hand", "part", "place", "case",
        "week", "company", "system", "program", "question", "work",
        "government", "number", "night", "point", "home", "water", "room",
        "mother", "area", "money", "story", "fact", "month", "lot", "right",
        "study", "book", "eye", "job", "word", "business", "issue", "side",
        "kind", "head", "house", "service", "friend", "father", "power",
        "hour", "game", "line", "end", "member", "law", "car", "city",
        "community", "name", "president", "team", "minute", "idea", "kid",
        "body", "information", "back", "parent", "face", "others", "level",
        "office", "door", "health", "person", "art", "war", "history",
        "party", "result", "change", "morning", "reason", "research", "girl",
        "guy", "moment", "air", "teacher", "force", "education"]
```

```
# https://www.espressoenglish.net/100-most-common-english-verbs/
verbs = ["be", "have", "do", "say", "go", "can", "get", "would", "make",
        "know", "will", "think", "take", "see", "come", "could", "want",
        "look", "use", "find", "give", "tell", "work", "may", "should",
        "call", "try", "ask", "need", "feel", "become", "leave", "put",
        "mean", "keep", "let", "begin", "seem", "help", "talk", "turn",
        "start", "might", "show", "hear", "play", "run", "move", "like",
        "live", "believe", "hold", "bring", "happen", "must", "write",
        "provide", "sit", "stand", "lose", "pay", "meet", "include",
        "continue", "set", "learn", "change", "lead", "understand", "watch",
        "follow", "stop", "create", "speak", "read", "allow", "add", "spend",
        "grow", "open", "walk", "win", "offer", "remember", "love",
        "consider", "appear", "buy", "wait", "serve", "die", "send", "expect",
        "build", "stay", "fall", "cut", "reach", "kill", "remain"]
```

BREAK

Python keywords are the following:

False, None, True, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

Defining a Function

You are already familiar with the print() and len() functions from the previous sessions. Python provides several built-in functions like these, but you can also write your own functions.

If you need to perform that task multiple times throughout your program, you do not need to type all the code for the same task again and again; you just call the function dedicated to handling that task, and the call tells Python to run the code inside the function. You will find that using functions makes your programs easier to write, read, test, and x.

```
def print_sentence():
    """Display the first sentence of Moby Dick"""

    fst_sentence = 'Call me Ishmael.'
    print(fst_sentence)

print_sentence()
```
Call me Ishmael.

This example shows the simplest structure of a function. The first line uses the keyword def to inform the Python interpreter, that you are defining a function. This is the function definition, which tells the interpreter the name of the function and, if applicable, what kind of information the function needs to do its job.

Arguments and Parameters

In the following example, modify_sentence(name) requires a value for the variable name. Once we called the function and gave it the information -a person's name-, it prints a modified sentence.

```
def modify_sentence(name):
    """Display a modified first sentence of Moby Dick
:param name: str Name to insert in the sentence."""

    fst_sentence = 'Call me ' + name + '.'
    print(fst_sentence)

modify_sentence('Ahab')
```

Call me Ahab.

Default Values

When writing a function, you can define a default value for each parameter. If an argument for a parameter is provided in the function call, Python uses the argument value. If not, it uses the parameter's default value. So when you define a default value for a parameter, you can exclude the corresponding argument you would usually write in the function call. Using default values can simplify your function calls and clarify the ways in which your functions are typically used.

OBS! When you use default values, any parameter with a default value needs to be listed after all the parameters that do not have default values. This allows Python to continue interpreting positional arguments correctly.

Equivalent Function Calls

Because positional arguments, keyword arguments, and default values can all be used together, often you will have several equivalent ways to call a function. Consider the following:

```
def apply_division(dividend, divisor=2):
    result = dividend / divisor
    print(result)

apply_division(5)
apply_division(dividend=5)

apply_division(5, 4)
apply_division(dividend=5, divisor=4)
apply_division(divisor=4, dividend=5)
```

Return Values

A function does not always have to display its output directly. Instead, it can process some data and then return a value or set of values. The value the function returns is called a return value. The return statement takes a value from inside a function and sends it back to the line that called the function. Return values allow you to move much of your program's grunt work into functions, which can simplify the body of your program.

A function can return any kind of value you need it to, including more complicated data structures like lists and dictionaries.

```
def modify_sentence(name):
    """Construct a modified first sentence of Moby Dick
        :param name: str Name to insert in the sentence.
        :return: list
        The modified first sentence as a list of words."""

    fst_sentence = 'Call me ' + name + '.'
    return fst_sentence.split()

result = modify_sentence('Ahab’)
Print(result)

['Call', 'me', 'Ahab.']
```

Passing an Arbitrary Number of Arguments

Sometimes you won't know ahead of time how many arguments a function needs to accept. Fortunately, Python allows a function to collect an arbitrary number of arguments from the calling statement.

```
def hire_crew(*sailors):
    """Print the list of hired crew members."""
    for sailor in sailors:
        print('- ' + sailor)

hire_crew('Ahab')
hire_crew('Ahab', 'Ishmael', 'Queequeg')
```

```
- Ahab
- Ahab
- Ishmael
- Queequeg
```

Mixing Positional and Arbitrary Arguments

If you want a function to accept several different kinds of arguments, the parameter that accepts an arbitrary number of arguments must be placed last in the function definition. Python matches positional and keyword arguments first and then collects any remaining arguments in the final parameter.

```
def hire_crew(amount, *sailors):
    """Checks the list of hired crew members."""
    members = []
    if amount != len(sailors):
        return False
    for sailor in sailors:
        members.append(sailor)
    return members

hire_crew(1, 'Ahab')
hire_crew(3, 'Ahab', 'Ishmael', 'Queequeg')
```

```
['Ahab', 'Ishmael', 'Queequeg']
```

Using Arbitrary Keyword Arguments

Sometimes you will want to accept an arbitrary number of arguments, but you won't know ahead of time what kind of information will be passed to the function. In this case, you can write functions that accept as many key-value pairs as the calling statement provides. One example involves building user profiles: you know you will get information about a user, but you are not sure what kind of information you'll receive. The function build_profile() in the following example always takes in a first and last name, but it accepts an arbitrary number of keyword arguments as well.

```
def build_profile(first, last, **user_info):
"""Build a dictionary containing everything we know about a user."""
    profile = {}
    profile['first_name'] = first
    profile['last_name'] = last
    for key, value in user_info.items():
        profile[key] = value
    return profile

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
print(user_profile)
{'first_name': 'albert’, 'last_name': 'einstein', 'location': 'princeton',
'field': 'physics'}
```

The definition of build_profile() expects a first and last name, and then it allows the user to pass in as many name-value pairs as they want. The double asterisks before the parameter **user_info cause Python to create an empty dictionary called user_info and pack whatever name-value pairs it receives into this dictionary. Within the function, you can access the name-value pairs in user_info just as you would for any dictionary.

Lambda epression.

What is lambda expression in Python?

Python allows you to create anonymous function i.e function having no names using a facility called lambda function. lambda functions are small functions usually not more than a line. The result of the expression is the value when the lambda is applied to an argument.

Lambda is a tool for building anonymous functions.

Python lambda Functions

```
print(lambda x: x + 1)(2))
```

Because a lambda function is an expression, it can be named. Therefore you could write the previous code as follows:

```
add_one = lambda x: x + 1
Print(add_one(2))
```

The above lambda function is equivalent to writing this:

```
def add_one(x):
   return x + 1
```

```
full_name = lambda first, last: f"Full name: {first.title()} {last.title()}"

print(full_name('guido', "van rossum"))

print(full_name( "Guido", "Van Rossum"))
```

```
Full name: Guido Van Rossum
Full name: Guido Van Rossum
```

The lambda function assigned to full_name takes two arguments and returns a string interpolating the two parameters first and last. As expected, the definition of the lambda lists the arguments with no parentheses, whereas calling the function is done exactly like a normal Python function, with parentheses surrounding the arguments.

A lambda function can be a higher-order function by taking a function (normal or lambda) as an argument like in the following example:

```
high_ord_func = lambda x, func: x + func(x)
x = high_ord_func(5, lambda x: x * x)
print(x)
```

Python exposes higher-order functions as built-in functions or in the standard library. Examples include map(), filter(), functools.reduce(), as well as key functions like sort(), sorted(), min(), and max(). You'll use lambda functions together with Python higher-order functions in Appropriate Uses of Lambda Expressions.

Use of lambda() with filter()

The filter() function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence “sequence”, for which the function returns True. Here is a small program that returns the odd numbers from an input list:

```
# Python code to illustrate
# filter() with lambda()
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambda x: (x%2 != 0) , li))
print(final_list)
```

Output:

```
[5, 7, 97, 77, 23, 73, 61]
```

Use of lambda() with map()

The map() function in Python takes in a function and a list as argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item. Example:

```
# Python code to illustrate
# map() with lambda()
# to get double of a list.
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(map(lambda x: x*2 , li))
print(final_list)
```

Output:

[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]

Importing an Entire Module

To start importing functions, we first need to create a module. A module is a file ending in .py that contains the code you want to import into your program.

To call a function from an imported module, enter the name of the module you imported followed by the name of the function separated by a dot.

```
import random

for idx in range(0,10):
    dice = random.choice([1, 2, 3, 4, 5, 6])
    print(idx, dice)
```

Module Aliases

You can also provide an alias for a module name. Giving a module a short alias.

```
import random as r

r.choice([1, 2, 3, 4, 5, 6])
```

Importing All Functions in a Module

You can tell Python to import every function in a module by using the asterisk (*) operator.

The asterisk in the import statement tells Python to copy every function from the module into this program file. Because every function is imported, you can call each function by name without using the dot notation. However, it is best not to use this approach when you are working with larger modules that you did not write yourself: if the module has a function name that matches an existing name in your project, you can get some unexpected results. Python may see several functions or variables with the same name, and instead of importing all the functions separately, it will overwrite the functions.

The best approach is to import the function or functions you want, or import the entire module and use the dot notation. This leads to clear code that is easy to read and understand.

from module_name import *

Importing Specific Functions

```
from module_name import function_0, function_1, function_2

from random import choice

choice([1, 2, 3, 4, 5, 6])
```

The following exercise is optional and below you can find the solution.

Write a function that iterates through a list and prints each value without using a loop. That is, write a recursive function to print the values of a list.

```
def print_each(my_list):
    print((my_list[0]))
    my_list.remove(my_list[0])
    if(len(my_list) > 0):
        print_each(my_list)

print_each([1,3,5,7])
```

MANDATORY

A solution strategy must be made for the project based on the following. Furthermore, it should be discussed with your review group.

Identifying the Problem:

- What do You want the Software to Do?.

Creating Your Wish List:

Devising a Solution:

- Understanding the Problem.
- Defining Acceptable Results.
- Breaking Down the Solution into Steps.
- Organizing the Tasks into Steps.
- Testing the Design and code.

Documenting the Solution:

- A tentative outline should be made for the 6 pages.