Lecture 1

Welcome to MUH101

Burkay Genç, Ahmet Selman Bozkır, and Selma Dilek

20/02/2024

Today

- course Info
- what is computation
- python basics
- mathematical operations
- python variables and types

Course Info

- **Subject** : Learn Python *a programming language*
- **Classes** :
  - All classes are face-to-face
  - All off-class communication will be on *Hacettepe University Digital Learning Platform (HADİ)*
    - hadi website
    - You have to follow hadi daily for new announcements

Course Info

- **Grading**
  - 60%: Midterm (2x)
    - Midterms format will be announced during the semester
  - 40%: Final Exam (1x)

Course Info

- You are **strictly forbidden to cooperate** on exams
- You are **strictly forbidden to search for answers on the Internet**
- You are **strictly forbidden to copy/paste materials from the Internet**

COURSE POLICIES

- Cooperation
  - you may cooperate with your friends to **understand the course material**
  - all exam work needs to be individual
    - cheaters will be **harshly** penalized -> **-100**, **F3**, **disciplinary committee**

FAST PACED COURSE

- Position yourself to succeed!
  - do practice early
  - do not skip lectures
- New to programming? *PRACTICE. PRACTICE? PRACTICE!*
  - can’t passively absorb programming as a skill
  - download code before lecture and follow along
  - don’t be afraid to try out Python commands!

TOPICS

- represent knowledge with **data structures**
- **iteration and recursion** as computational metaphors
- **abstraction** of procedures and data types
- **organize and modularize** systems using object classes and methods
- [maybe] different classes of **algorithms**, searching and sorting
- [maybe] **complexity** of algorithms

WHERE TO GET PYTHON

- You can download your own copy, it is free
  - Download Python *recommended*
- More advanced tasks (data science, machine learning)
  - Download Anaconda
  - Not recommended at this level
- You can use online compilers/IDEs
  - Google Colab *recommended will be used in course*
  - Jupyter Notebook *recommended*
  - Python Fiddle
  - PyFiddle
  - Programiz
- Use a Linux distribution such as Ubuntu, Manjaro etc.
  - Python comes built in

PYTHON IDE

- What is an **IDE**?
  - An **I**ntegrated **D**evelopment **E**nvironment allows you to write programs in a programming language and provides extra tools to help the process
    - Syntax highlighting
    - Code completion
    - Bracket completion/matching
    - Debugging
    - Profiling
    - more…
- PyCharm is a very powerful and popular IDE for Python
- IDLE comes built-in with any Python distribution
- Eric, Spyder, Eclipse+PyDev
- Sublime Text, *VS Code*, Atom, etc. are generic editors that can be extended with Python capabilities

WHAT DOES A COMPUTER DO

- Fundamentally:
  - performs **calculations**
    a billion calculations per second!
  - **remembers** results
    100s of gigabytes of storage!
- What kinds of calculations?
  - **built-in** to the language
  - ones that **you define** as the programmer
- computers only know what you tell them

TYPES OF KNOWLEDGE

- **declarative knowledge** is *statements of fact*.
  - someone will win a Trophy before class ends
- **imperative knowledge** is a *recipe* or “how-to”.
  1. Students sign up for lottery
  2. Burkay opens his IDE
  3. Burkay chooses a random number between 1^{st} and n^{th} responder
  4. Burkay finds the number in the responders sheet. Winner!

A NUMERICAL EXAMPLE

- square root of a number x is y such that $y * y = x$
- recipe for deducing square root of a number x

  1. Start with a **guess**, g
  2. If $g * g$ is **close enough** to x, stop and say g is the answer
  3. Otherwise make a **new guess** by averaging g and x/g
  4. Using the new guess, **repeat** process until close enough

g	$g * g$	x/g	$(g + x/g)/2$
3	9	16/3	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

WHAT IS A RECIPE

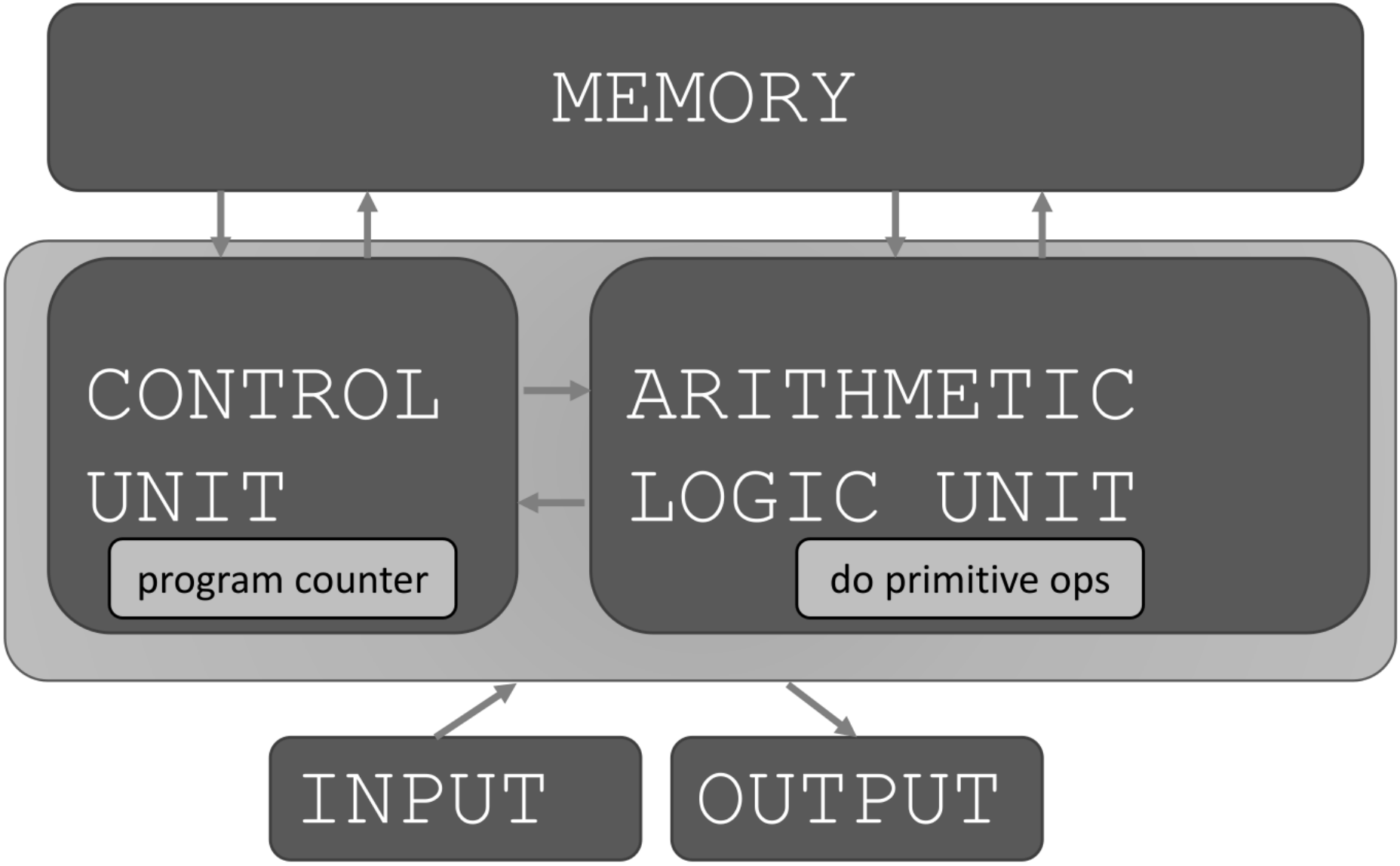
1. sequence of simple **steps**
2. **flow of control** process that specifies when each step is executed
3. a means of determining **when to stop**

1 + 2 + 3 = an **algorithm**!

COMPUTERS ARE MACHINES

- how to capture a recipe in a mechanical process
- **fixed program** computer
  - calculator
- **stored program** computer
  - machine stores and executes instructions

BASIC MACHINE ARCHITECTURE



Basic machine architecture

STORED PROGRAM COMPUTER

- sequence of **instructions stored** inside computer
  - built from predefined set of primitive instructions
    - arithmetic and logic
    - simple tests
    - moving data
- special program (interpreter) **executes each instruction in order**
  - use tests to change flow of control through sequence
  - stop when done

BASIC PRIMITIVES

- Turing showed that you can **compute anything** using 6 primitives
- modern programming languages have more convenient set of primitives
- we can also **create new primitives**
- anything computable in one language is computable in any other programming language
  - Python == Java == C == Pascal == C++ == C#
  - They only differ in **ease** of doing something

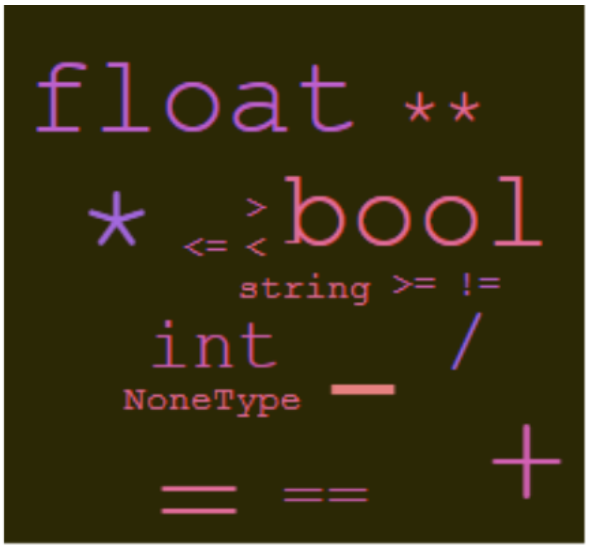
CREATING RECIPES

- a programming language provides a set of primitive **constructs**
  - 2, 4.8, ‘a’, “burkay”, TRUE, +, -, *, …
- **expressions** are complex but legal combinations of primitives in a programming language
  - 2 + 4.7
  - TRUE && FALSE
  - 3.0 + 2 / 5.4
- expressions and computations have **values** and meanings in a programming language

ASPECTS OF LANGUAGES

- **primitive constructs**
  - English: words
  - programming language: numbers, strings, simple operators

Word Cloud copyright Michael Twardos, All Right Reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.



Word Cloud copyright unknown, All Right Reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.

ASPECTS OF LANGUAGES

- **syntax**
  - English:

“cat dog boy”	not syntactically valid
“cat hugs boy”	syntactically valid

  - programming language:

“hi”5	not syntactically valid
3.2 * 5	syntactically valid

ASPECTS OF LANGUAGES

- **static semantics** is which syntactically valid strings have meaning
  - English:

"I are hungry"	syntactically valid but static semantic error

  - programming language:

3.2*5	syntactically valid
3+"hi"	static semantic error

ASPECTS OF LANGUAGES

- **semantics** is the meaning associated with a syntactically correct string of symbols with no static semantic errors
  - English: can have many meanings `"Flying planes can be dangerous"`
  - programming languages: have only one meaning but may not be what programmer intended
  - Trying to print the square of 5:

```
x = 5
print(x * 2)
```

```
## 10
```

- Oops! The program runs, because there are **no static semantic errors**. But it doesn't do the *intended* thing. Here is a corrected version:

```
x = 5
print(x ** 2)
```

```
## 25
```

WHERE THINGS GO WRONG

- **syntactic errors**
  - common and easily caught
- **static semantic errors**
  - some languages check for these before running program
  - can cause unpredictable behavior
- no static semantic errors but **different meaning than what programmer intended**
  - program crashes, stops running
  - program runs forever
  - program gives an answer but different than expected

EXAMPLES

- syntactic errors

```
print(3.a)
```

```
SyntaxError: invalid syntax
```

```
a = 5
3a + 2
```

```
SyntaxError: invalid syntax
```

```
a - 3'a'
```

```
SyntaxError: invalid syntax
```

- static semantic errors

```
3 + 'a'
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

PYTHON PROGRAMS

- a **program** is a sequence of definitions and commands
  - definitions *evaluated*
  - commands *executed* by Python interpreter in a shell
- **commands** (statements) instruct interpreter to do something
- can be typed directly in a *shell* or stored in a *file* that is read into the shell and evaluated

OBJECTS

- programs manipulate **data objects**
- objects have a **type** that defines the kinds of things programs can do to them
  -  Burkay is a human so he can walk, speak English, etc.
  -  Chewbacca is a wookie so he can walk, “mwaaarhrhh”, etc.
- objects are
  - scalar (cannot be subdivided)
  - non-scalar (have internal structure that can be accessed)

SCALAR OBJECTS

- `int` – represent **integers**, ex. `5`
- `float` – represent **real numbers**, ex. `3.27`
- `bool` – represent **Boolean** values `True` and `False`
- `NoneType` – **special** and has one value, `None`
- can use `type()` to see the type of an object

```
type(5)
```

```
## <class 'int'>
```

```
type(3.0)
```

```
## <class 'float'>
```

NON-SCALAR OBJECTS

- Strings are non-scalar objects. They have internal structures.

```
name = "burkay"
print(name[2:4])
```

```
## rk
```

- We can construct new non-scalar objects.
- **Object oriented programming** is the art of programming using non-scalar objects.

TYPE CONVERSIONS (CAST)

- We can **convert object of one type to another**
  - Not all types are convertible!
- `float(3)` converts integer `3` to float `3.0`
- `int(3.9)` truncates float `3.9` to integer `3`

```
float(3)
```

```
## 3.0
```

```
int(3.9)
```

```
## 3
```

```
int("burkay")
```

```
## ValueError: invalid literal for int() with base 10: 'burkay'
```

PRINTING TO CONSOLE

- to show output from code to a user, use `print` command

```
print("Hello World!")
```

```
## Hello World!
```

```
print(3 + 2)
```

```
## 5
```

```
print("My age is", 41)
```

```
## My age is 41
```

EXPRESSIONS

- **combine objects and operators** to form expressions
- an expression has a **value**, which has a type
- syntax for a simple expression
  `<object> <operator> <object>`

OPERATORS ON ints and floats

- `i+j` → the **sum**
- `i-j` → the **difference**
- `i*j` → the **product**
- `i/j` → the **division**
- For the sum, the difference and the product, if both objects are integers then the result is an integer. If one or both are floats, then the result is a float.
- For the division the result is always a float.
- `i%j` → the **remainder** when `i` is divided by `j`
- `i**j` → `i` to the power of `j`

OPERATORS ON ints and floats

```
3 + 5
```

```
## 8
```

```
3 - 5
```

```
## -2
```

```
3 * 5
```

```
## 15
```

```
3 / 5
```

```
## 0.6
```

```
32 % 5
```

```
## 2
```

```
3 ** 4
```

```
## 81
```

SIMPLE OPERATIONS

- parentheses are used to tell Python to prioritize operations

```
3 * (2 + 5)
```

```
## 21
```

- **operator precedence** without parentheses
  - `**`
  - `*`
  - `/`
  - `+` and `-` executed left to right, as appear in expression

```
3 * 2 + 5
```

```
## 11
```

BINDING VARIABLES AND VALUES

- equal sign is an **assignment** of a value to a variable name

```
# variable  =  value
pi        = 3.14159

pi_approx = 22/7
```

- value stored in computer memory
- an assignment binds variable name to value
- retrieve value associated with variable name by invoking the name, by typing `pi`

```
pi
```

```
## 3.14159
```

```
pi_approx
```

```
## 3.142857142857143
```

MULTIPLE ASSIGNMENTS

- you can assign multiple values to variables at once

```
a, b = 3, 5
a
```

```
## 3
```

```
b
```

```
## 5
```

VARIABLE NAMING

- it is important to use clear and understandable names for variables
- also, using parenthesis in expressions helps with code readability

```
a, b, c = 5, 8, 3.14
d = c * a ** 2 * b
```

vs.

```
r, h, pi = 5, 8, 3.14
V_cyl = pi * (r ** 2) * h
```

$$V_{cyl} = \pi r^2 h$$

COMMENTS

- you can enrich your code by adding comments
  - to add a comment, start a line with `#`
  - lines starting with `#` are ignored by Python

```
# Radius, height and pi are defined
r, h, pi = 5, 8, 3.14

# The volume of the cylinder is computed
# Volume is equal to height times the base area
V_cyl = pi * (r ** 2) * h
```

ABSTRACTING EXPRESSIONS

- why **give names** to values of expressions?
- to **reuse names** instead of values
- easier to change code later

```
pi = 3.14159
radius = 2.2
area = pi * (radius ** 2)
print(area)
```

```
## 15.205295600000001
```

PROGRAMMING vs MATH

- in programming, variables do not get automatically updated

```
pi = 3.14159
radius = 2.2
# area of circle
area = pi * (radius**2)
print(area)
```

```
## 15.205295600000001
```

```
radius = radius + 1
print(radius)
```

```
## 3.2
```

```
print(area)
```

```
## 15.205295600000001
```

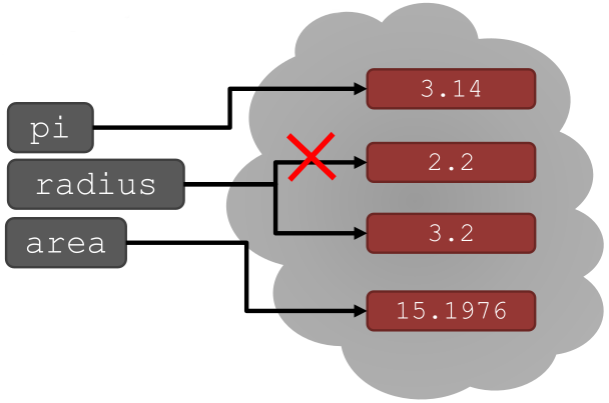
```
area = pi * (radius**2)
print(area)
```

```
## 32.169881600000004
```

CHANGING BINDINGS

- can re-bind variable names using new assignment statements
- previous value may still be stored in memory but lost the handle for it
- value for area does not change until you tell the computer to do the calculation again

```
pi = 3.14
radius = 2.2
area = pi * (radius**2)
radius = radius + 1
```



Exercise

1. What is printed when the code snippet below is executed?

```
type(5)
print(3.0 - 1)
```

1. <class ‘int’>
2. 2.0
3. <class ‘int’> then 2.0
4. nothing

Try the code: https://www.tutorialspoint.com/python/online-python-compiler.php

Exercise

1. Which expression is allowed in Python?

1. x + y = 2
2. x*x = 2
3. 2 = x
4. xy = 2

Exercise

1. What is printed when the code snippet below is executed?

```
a = 6
b = 7
c = 1
total = a + b + c
print(total)
c += 1 # Same as: c = c + 1
print(total)
```

1. 14 then 14
2. 14 then 15
3. 14
4. 15

Copyright Information