WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Principles of Software Programming: Recap

Svitlana Vakulenko, MSc.

WS 2017

Acknowledgement: slides adopted from Dr. Jürgen Umbrich GZP SS 2014 lecture slides

This Episode

- 13:00-16:00
- Jupyter notebooks
- Forum
- Sample exam
  - Data types
  - Functions
  - OOP
- Practice

Exam

1. **Basics:** algorithm, compiler/interpreter, variable, operator, data types, arrays

2. **Control flow:** conditions, loops

3. **Object-oriented programming:** Class, Object, Constructor, Abstraction, Inheritance, Encapsulation, Polymorphism

Extra:

1. Exceptions
2. Data structures: Stack, Queue, Map (Dictionary), Tree

Sample exam

- 3 ex. x 20 p. = 60 p.
- + (8 p. + 7 p.) = 75 p.
  - 1. Theory
    - 1c) Program output
  - 2. Write a program x 2
  - 3. OOP
- https://learn.wu.ac.at/dotlrn/classes/gdp/res/concept-space/index?expand=23073075

Primitive Data Types (Java)

- **byte** = 8 bits
- **short**: 2 bytes = 16 bits
- **int**: 4 bytes = 32 bits
- **long**: 8 bytes = 64 bits

- [-128; 127]
- [-32,768; 32,767]
- 2,147,483,647
- 9223372036854775807

- **float**: 4 bytes = 32 bits
- **double**: 8 bytes = 64 bits

- e.g. 3.141

- **char**: 2 bytes = 16 bits
- **boolean**: 1 bit

- Unicode, e.g. ‘A’, 65
- false, true

https://en.wikibooks.org/wiki/Java_Programming/Primitive_Types

(Un)signed types

- **signed** data types can have positive and negative values
- **unsigned** can only represent non-negative numbers

- byte = 8 bits
- signed byte: [-128; 127]
- unsigned byte: [0; 255]

https://en.wikipedia.org/wiki/Signedness

http://ee.hawaii.edu/~tep/EE160/Book/chap5/subsection2.1.1.1.html

Data Types (Python)

- life_is_good = True
- hamsters_are_evil = False
- size_of_shoes = 42
- earth_population = 7000000000
- pi = 3.14159265359
- chinese_hi = “嗨"
- my_new_book = None

http://pythonvisually.com/ebook/primitive-data-types.html

Typing

- **static** typed language (Java)
  - have to initialise variables
  - type checking is done at compile-time

```
int num;
num = 5;
```

- **dynamic** (Python)
  - no declaration of variables before they're used
  - type checking is done at run-time

```
num = 5
```

https://www.sitepoint.com/typing-versus-dynamic-typing/

https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html

Spelling mistake

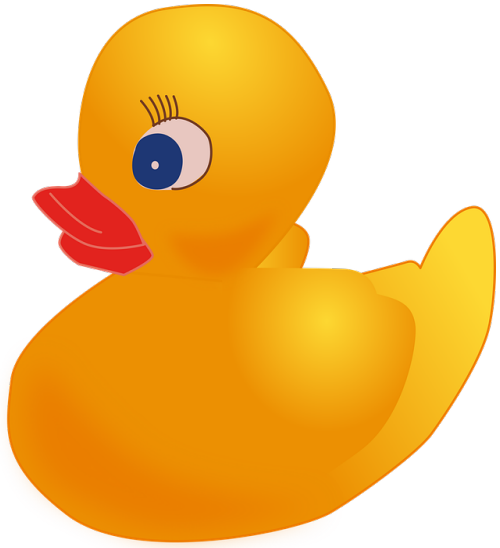
```
my_variable = 10
while my_variable > 0:
        i = foo(my_variable)
        if i < 100:
                my_variable++
        else
                my_varaible = (my_variable + i) / 10
```

https://www.sitepoint.com/typing-versus-dynamic-typing/

Duck typing

- "If it walks like a duck and it quacks like a duck, then it must be a duck."

```
class Sparrow:
    def fly(self):
        print("Sparrow flying")

class Airplane:
    def fly(self):
        print("Airplane flying")

class Whale:
    def swim(self):
        print("Whale swimming")

def lift_off(entity):
    entity.fly()

sparrow = Sparrow()
airplane = Airplane()
whale = Whale()

lift_off(sparrow) # prints `Sparrow flying`
lift_off(airplane) # prints `Airplane flying`
lift_off(whale) # Throws the error `'Whale' object has no attribute 'fly'`
```



https://pixabay.com/en/bath-duck-rubber-toy-verbs-2022661/

https://en.wikipedia.org/wiki/Duck_typing

Type conversion

a = int(“34")

b = float(“3.1415926”)

```
>>> i = 10
>>> float(i)
10.0

>>> f = 14.66
>>> int(f)
14

>>> i = 100
>>> str(i)
“100”
```

https://stackoverflow.com/questions/12161402/why-does-python-dont-have-implicit-conversions

http://www.informit.com/articles/article.aspx?p=459269&seqNum=7

Explicit is better than implicit

```
print(“Hello" + str(1))
```

```
System.out.print("Hello " + 1);
```

Function

- block of re-usable code to perform specific tasks
- (optional) parameters (arguments)
- overloading

```
def defArgFunc(empname, emprole = "Manager"):
    print ("Emp Name: ", empname)
    print ("Emp Role ", emprole)

defArgFunc(empname="Nick")
defArgFunc(empname="Tom",emprole = "CEO")
```

https://www.codementor.io/kaushikpal/user-defined-functions-in-python-8s7wyc8k2

Return statement

```
def add(value1, value2):
    return value1 + value2

result = add(3, 5)
print(result)
```

```
def profile():
    name = "Danny"
    age = 30
    return (name, age)

profile_data = profile()
print(profile_data[0])
print(profile_data[1])
```

http://book.pythontips.com/en/latest/global_&_return.html

Instantiating

```
class Point:
    """ Point class represents and manipulates x,y coords. """

    def __init__(self):
        """ Create a new point at the origin """
        self.x = 0
        self.y = 0


p = Point()           # Instantiate an object of type Point
q = Point()           # Make a second point

print(p.x, p.y, q.x, q.y)
```

http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html

Class & instance variables

```
class Shark:

    animal_type = "fish"
    location = "ocean"

    def __init__(self, name, age):
        self.name = name
        self.age = age

sammy = Shark("Sammy", 5)
print(sammy.name)
print(sammy.location)

stevie = Shark("Stevie", 8)
print(stevie.name)
print(stevie.animal_type)
```

https://www.digitalocean.com/community/tutorials/understanding-class-and-instance-variables-in-python-3

Inheritance

- is-a relationship

```
class Instrument:
    pass

class StringInstrument(Instrument):
    def play_melody(self):
        print("la-la-la")

class PercussionInstrument(Instrument):
    def keep_the_beat(self):
        print("tam-tam-tam")
```

Multiple inheritance

```
class Instrument:
    pass

class StringInstrument(Instrument):
    def play_melody(self):
        print("la-la-la")

class PercussionInstrument(Instrument):
    def keep_the_beat(self):
        print("tam-tam-tam")


class Piano(StringedInstrument, PercussionInstrument):
    pass


p = Piano()
p.play_melody()
p.keep_the_beat()
```

https://www.journaldev.com/14623/python-multiple-inheritance
https://www.python-course.eu/python3_multiple_inheritance.php

Method overriding

```
class Dog:
    def bark(self):
        print "WOOF"

class CrazyDog( Dog ):
    def bark( self ):
        print "WoOoOoF!!"

myDog= Dog()
myDog.bark() # WOOF

boby = CrazyDog()
boby.bark()
```

https://stackoverflow.com/questions/394770/override-a-method-at-instance-level

Super constructor

```
class Mammal():
  def __init__(self, mammalName):
    print(mammalName, 'is a warm-blooded animal.')

class Dog(Mammal):
  def __init__(self):
    print('Dog has four legs.')
    super().__init__('Dog')

d1 = Dog()
```

https://www.programiz.com/python-programming/methods/built-in/super

Composition

- has-a relationship

```
class RoboticArm():
    def __init__(self, length):
        self.length = length

class MarsRover():
    def __init__(self, name):
        self.name = name
        self.arm = RoboticArm(2.1)


if __name__ == "__main__":
    curiosity = MarsRover("Curiosity")
```

https://en.wikipedia.org/wiki/Curiosity_(rover)

https://en.wikipedia.org/wiki/Mars_rover

Ex.1: Happy Birthday!

- ask user for name and age
- show the year when user will turn 100 years old

https://pixabay.com/en/cane-elder-elderly-grandfather-1293369/

Job interview

- **Coding:** write some simple code, with correct syntax
- **OO design:** basic concepts, model a simple problem
- **Scripting and regexes:** find the phone numbers
- **Data structures:** arrays, hashtables, trees, graphs
- **Bits and bytes:** binary numbers

Example Coding/Engineering Interview

https://www.youtube.com/watch?v=ko-KkSmp-Lk

https://sites.google.com/site/steveyegge2/five-essential-phone-screen-questions

Bits and bytes

- Computers don't have ten fingers; they have one
- most significant (sign) bit

0 0 0 1 numerical value 2^0
0 0 1 0 numerical value 2^1
0 1 0 0 numerical value 2^2
1 0 0 0 numerical value 2^3

- 2^5
- 2^8 (byte = 8 bits, [0; 255], [-128; 127])
- 2^{10}
- 2^{16} (short: 16 bits, [-32.768; 32.767])

https://en.wikipedia.org/wiki/Binary_number

Operations

- bitwise logical
- NOT, AND, OR, XOR

```
NOT 0111
  = 1000
```

```
    0101
AND 0011
  = 0001
```

- bit shift (left/right)

```
00010111
00101110
```

```
10010111
11001011
```

```
00010111
01011100
```

https://en.wikipedia.org/wiki/Bitwise_operation

https://sites.google.com/site/steveyegge2/five-essential-phone-screen-questions

Number systems

- **octal** (base 8, octet, byte)
  - [0, 1, 2, 3, 4, 5, 6, 7]
  - 10, 11 ?

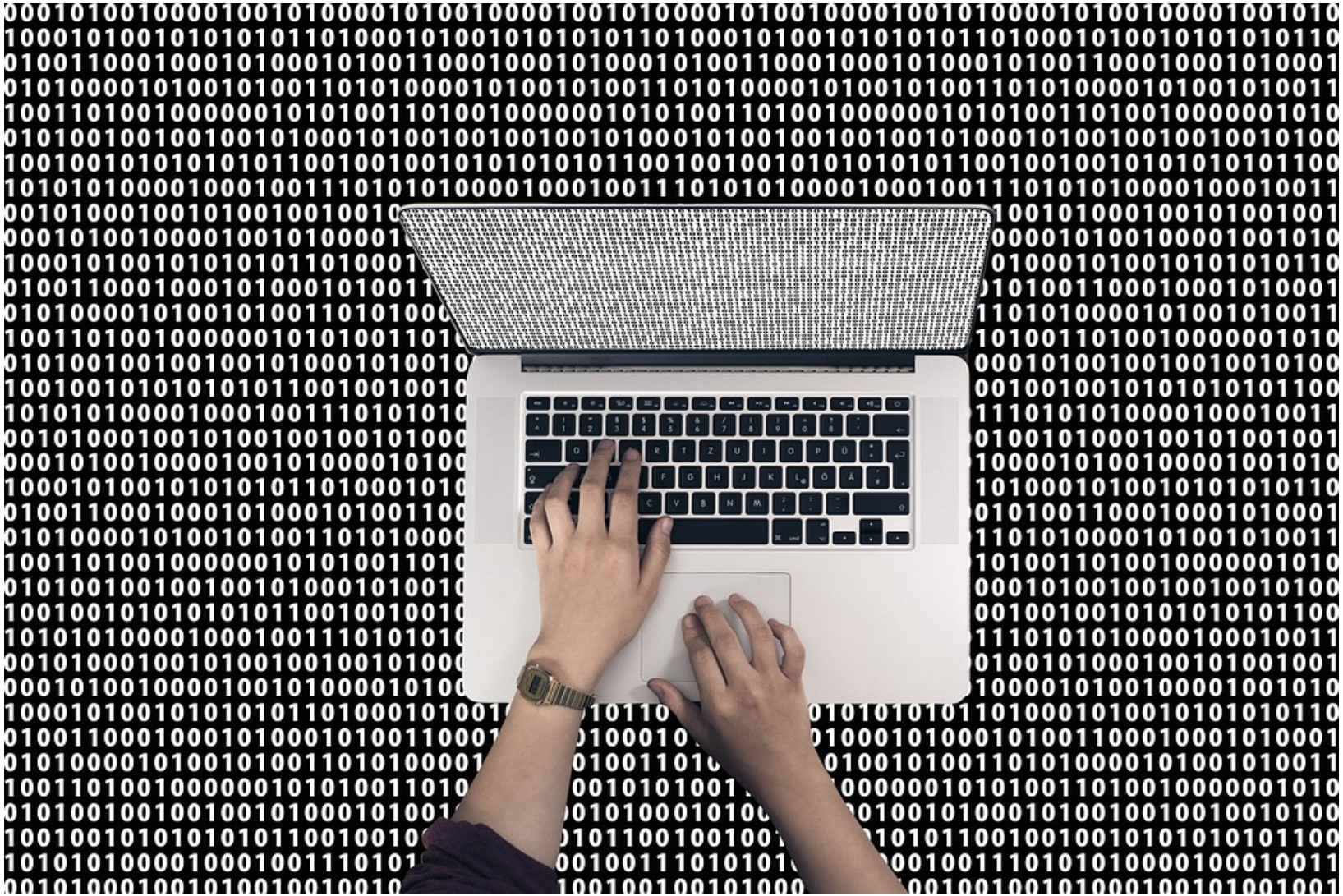
- **hexadecimal** (base 16, hex)
  - [0,…,9, A, B, C, D, E, F]
  - 10, FF (byte) ?

http://www.electronics-tutorials.ws/binary/bin_3.html

https://en.wikipedia.org/wiki/Hexadecimal

http://www.electronics-tutorials.ws/binary/bin_4.html

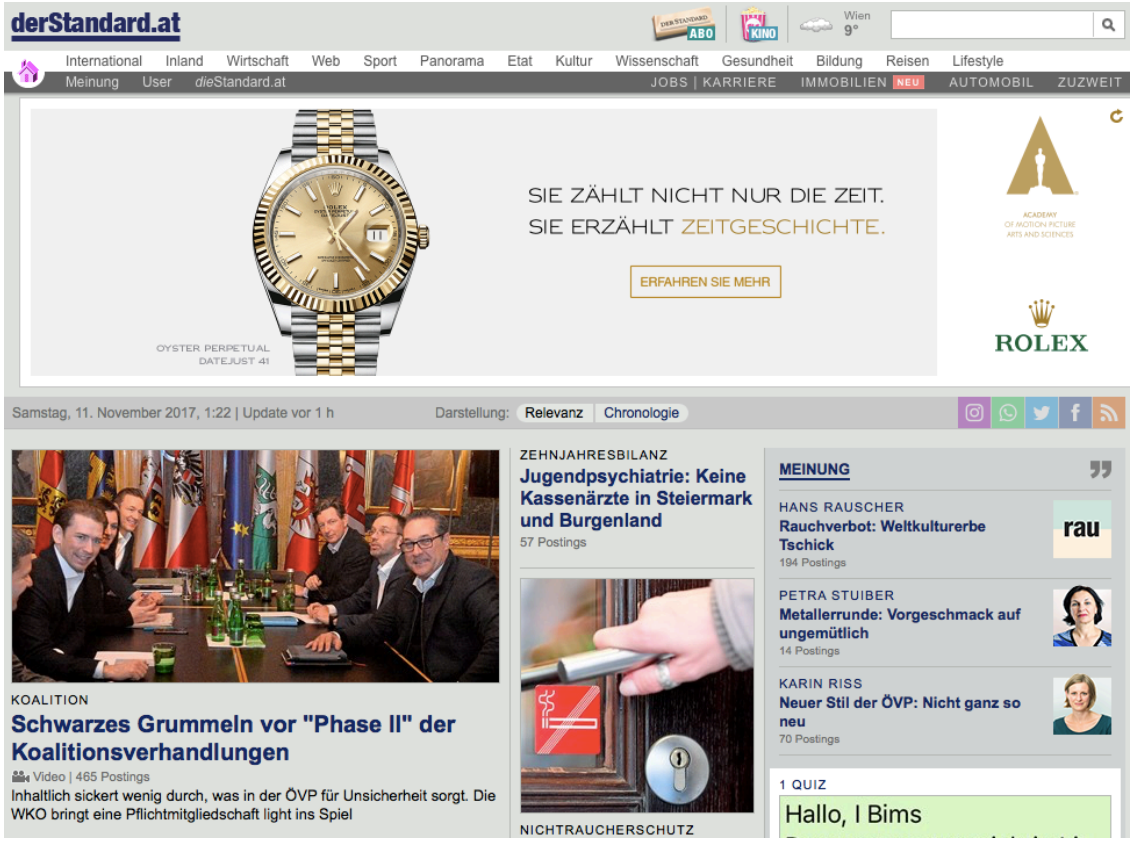
What is Programming?



https://pixabay.com/en/photos/computer%20science/

Project

- list all article titles from standard.at
- Beautiful Soup



http://www.practicepython.org/exercise/2014/06/06/17-decode-a-web-page.html

Resources

- MIT Course
- Python Tutor
- Interactive book

Web development

- Flask
- Django

Visual arts

- Processing
- Unity

Music

- Stanford Laptop Orchestra
- The DIY orchestra of the future
- Waves Vienna Hackathon

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Thank you!

svitlana.vakulenko@wu.ac.at

Schedule

	Topics	Dates
1	Course Overview, Introduction Python	Monday 10/30/17
2	Structured & Object-oriented paradigms	Friday 11/03/17
3	Data Structures: List, Set, Dictionary	Monday 11/06/17
4	Version Control, Project Structure	Wed 11/08/17
5	Files: Input/Output	Friday 11/10/17
6	Debugging: Exceptions, Assertions	Monday 11/13/17
7	Recap*	Wed 11/15/17
8	**Trees, Recursion, Sort&Search***	Friday 11/17/17

01:00 PM - 03:45 PM D2.0.031 Workstation-Raum
*01:00 PM - 04:00 PM