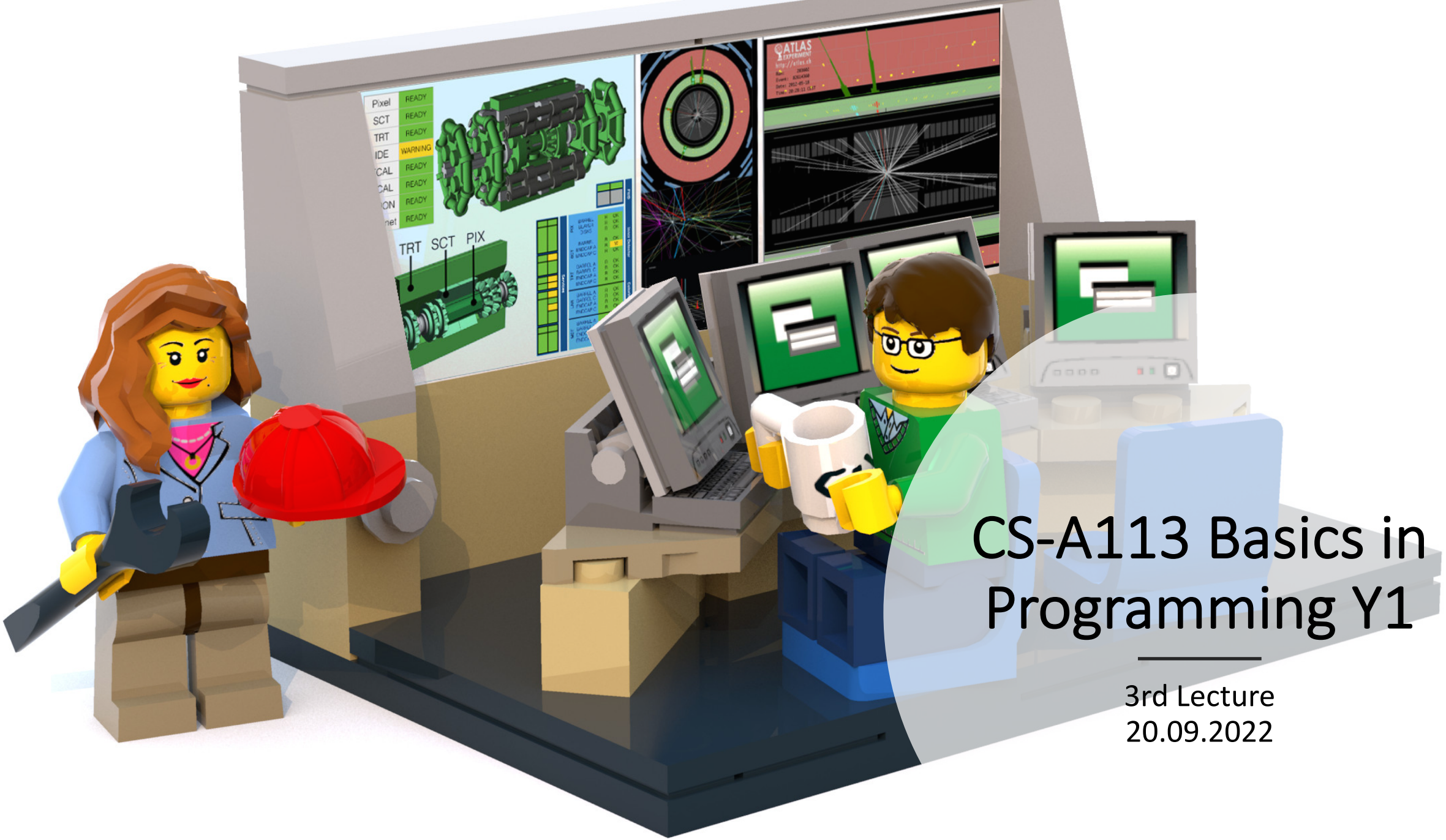
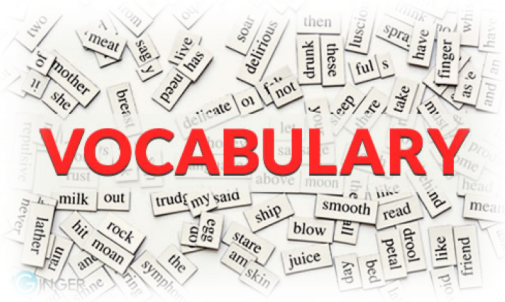


CS-A113 Basics in Programming Y1

3rd Lecture
20.09.2022

For-Loop and Range

Vocabulary and Format



Coding Style & Debugging 101

Topics Today

LOOPS

while and for

Loops Recap:

```
def superLoop():
    i = 1
    j = i
    while i < 5:
        print(i)
        j += 1
```

For-Loop with Range-Function

```
def superLoop1():
    for i in range(8):
        print(i)
```

```
0,1,2,3,4,5,6,7
```

```
def superLoop2():
    for i in range(0,8):
        print(i)
```

```
0,1,2,3,4,5,6,7
```

```
def superLoop2():
    for i in range(2,8):
        print(i)
```

```
2,3,4,5,6,7
```

for i in range(x,y) ->

i= x

i = x+1

i=x+2

…

i=y-1

Loops

```
def superLoop():
    myNumber = int(input(‘Say a number.\n’)):
    for i in range(myNumber)
        print(i)
```

Range in more Detail

range(5) = range(0,5,1)

range(x,y,z)

x,
x+z,
x+2z,
x+3z,
...
x+wz<y

We want to print all positive, odd numbers until 100.

```
def superLoop():
    for i in range(x,y,z):
        print(i)
```

Why Start from Zero?

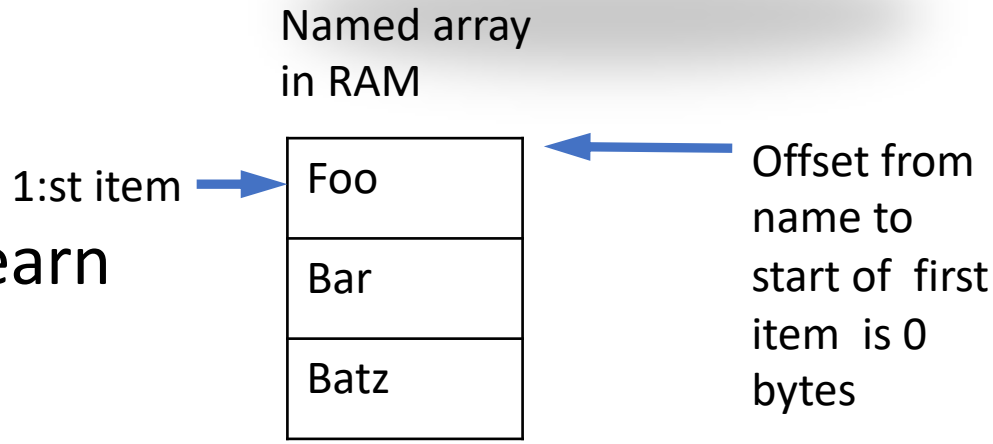
range(n) returns n numbers, which is desirable

But annoyingly from 0 to n-1

This is common custom in computing, caused by some lower level languages like C

Which is due to the memory offset from a named memory area to the first item

This is a convention and you just have to learn to live with it



Loops

```
def superLoop():
    for i in range(x,y,z):
        print(i)
```

We want all negative Numbers greater than -20 which are divisible by 3 without remainder

We want all numbers in a descending order between 20 and 0 that are divisible by 3 without remainder

Why both *while* and *for*?

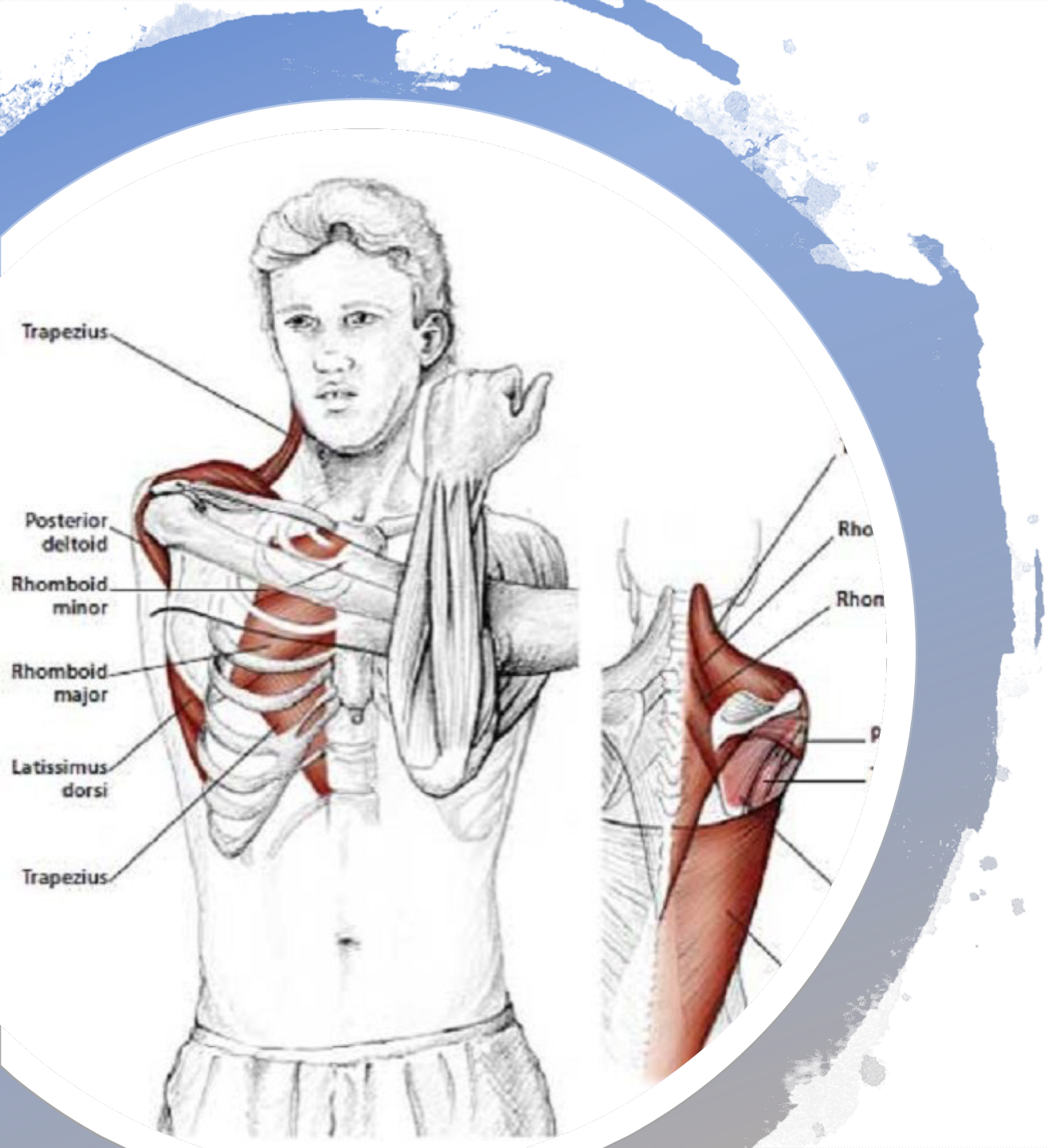
For is for processing a known data sequence

range(), list, set

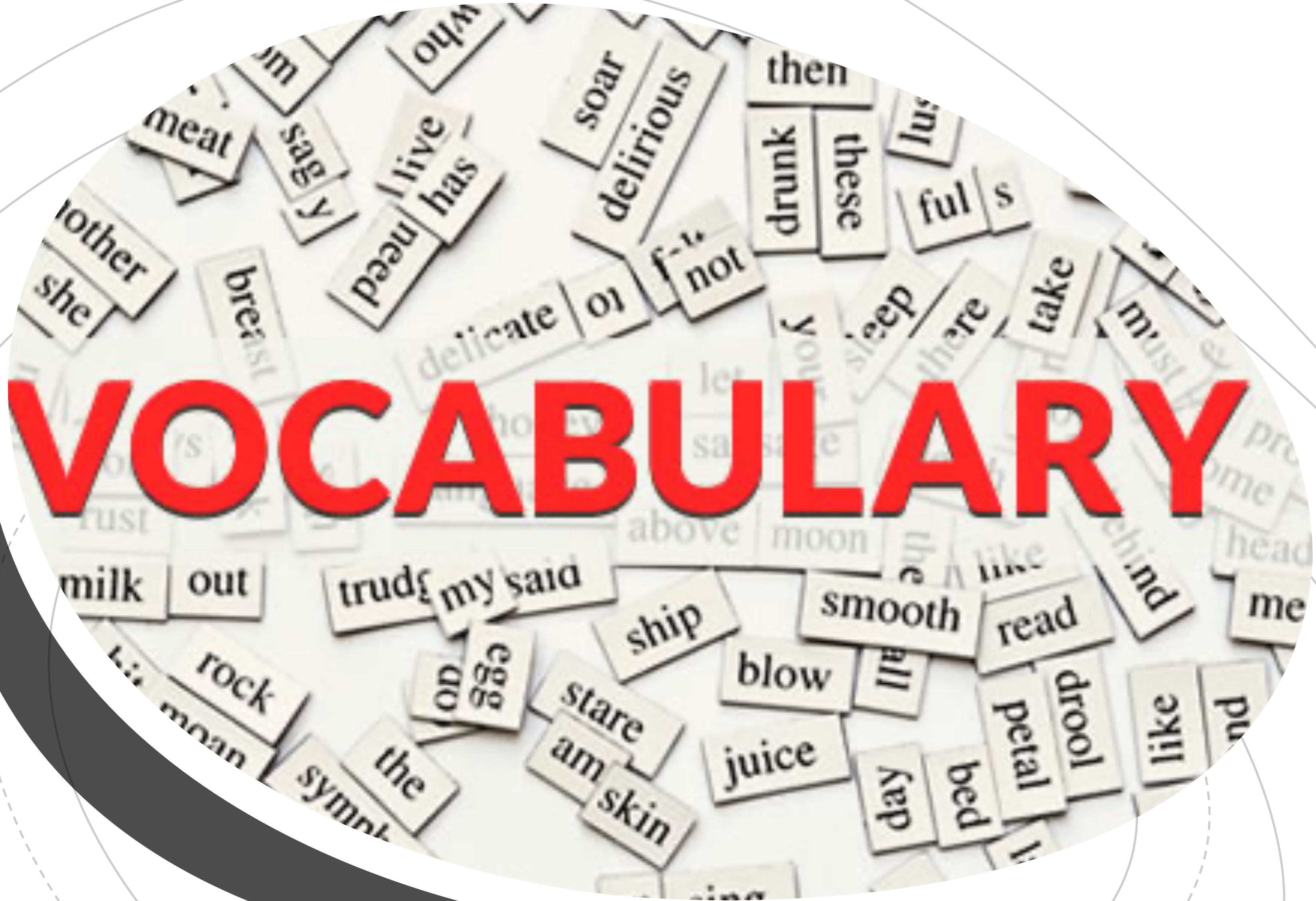
While is to repeat something until a condition is met

Even if one can often replace another, there are differences and knowing which to use makes code easier to understand

Also, there are implementation differences and nuances

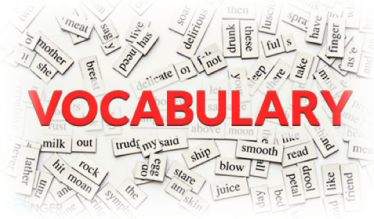


Break: Move your Shoulders


VOCABULARY
then
soar
delirious
meat
drunk
these
live
has
need
ful
s
she
take
not
delicate
must
above
moon
head
milk
out
my
said
ship
smooth
read
blow
rock
egg
stare
juice
am
skin
the
day
bed
petal
drool
like

and Format ;)

Incomprehensive, Non-Accurate List: Programming words



Syntax: ‘structure’ of the program e.g., for x in range(n):

Semantic: What the program actually does e.g., loop through 0 to n-1

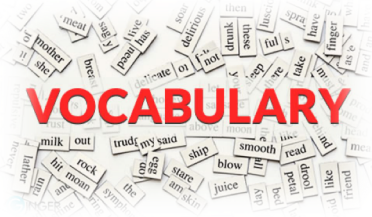
Algorithm: A way of solving a problem by methodological steps

Program: An instruction for the computer to follow

Input: What is given to a program

Output: What the program returns

Incomprehensive, Non-Accurate List Data words



String: one or several characters

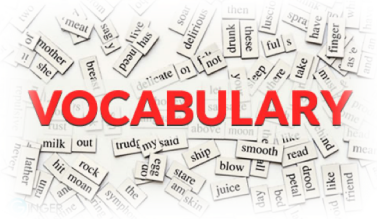
Integer: a ‘normal’ number

Float: a number that can have a decimal point

List: a list of elements with an order

Bool: a logical value, True or False

Incomprehensive, Non-Accurate List Working words



Comment: Something the computer ignores when running a program

IDE: Integrated Development Environment (e.g. PyCharm, Eclipse)

Python File: A file with the endin ”.py” should contain python code

Debugging: Correcting the code to do what one wants it to do

Formatting output

print(firstname, “earned“, salary, “Euros in“, month, “. “)

Alex earned 100.126735486649 Euros in September .

Joe earned 10.5 Euros in August .

Sally earned 2450.5999 Euros in August .

print(firstname+“ earned “+str(salary)+“ Euros in “+month+“. “)

Alex earned 100.126735486649 Euros in September.

Joe earned 10.5 Euros in August.

Sally earned 2450.5999 Euros in August.

We want an easy way to make output pretty, especially when printing lists of things

Format Strings with .format()

```
("{:} earned {:} Euros in {:}.").format(firstname,salary,month)
Alex earned 100.126735486649 Euros in September.

print(("{:15s} earned {:} Euros in {:}.").format(firstname,salary,month))
Alex            earned 100.126735486649 Euros in September.

print(("{:15s} earned {:7f} Euros in {:}.").format(firstname,salary,month))
Alex            earned 100.126735 Euros in September.

print(("{:15s} earned {:7.2f} Euros in {:}.").format(firstname,salary,month))
Alex            earned   100.13 Euros in September.

("{:15s} earned {:7.2f} Euros in {:>9s}.").format(firstname,salary,month)
Alex            earned   100.13 Euros in September.
Joe             earned    10.50 Euros in    August.
Sally           earned  2450.60 Euros in    August.
```

Using print() and Formatting

- We want to have nice output on screen
  - Data in justified columns
  - Proper number of decimals for a float
- Python has two methods for this
  - format() is the older (Python 2.6)
  - F-strings is nicer from Python 3.6 onwards
  - Both use the same formatting method
  - Quite a lot of details we don't get into

- Programming languages evolve and minor features get added on

Alternative f-strings

Since python 3.6 you can use f-strings (slightly less confusing)

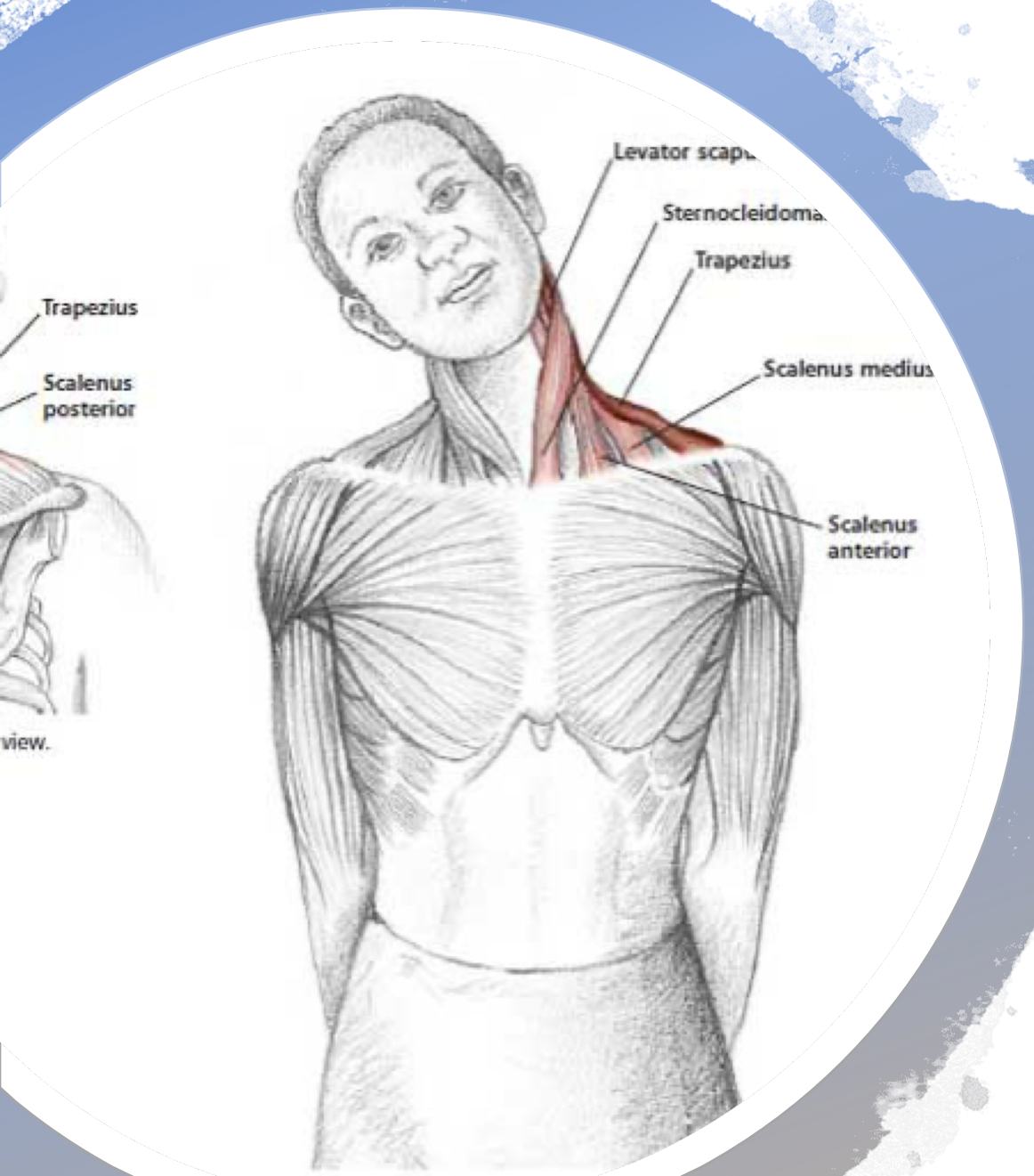
Same formatting system, but variables and values inside the string

```
number  = 5
factor  = 8
result  = number * factor
print(f"{number:3d} times {factor:3d} is {result:6d}")
 5 times   8 is     40

print(("{:3d} times {:3d} is {:6d}").format(number,factor,result))
```

Are all Programming Languages Same?

- Some things are found in almost all languages
  - For and while loops
  - If-else structure
  - Named variables and functions
  - These have some variation, but are fundamentally same
- Print formatting and pretty printing is very common
  - However, very much variation in implementation



Break:
Move your Neck!

Coding Style

We will Look at Your Style on Some Point!

Keep it clean ;)

Naming, Commenting, Structuring

- Programs are written:
  - For the computer to run mechanically
  - For other programmers to maintain and develop further
- Computers will eat anything
  - See the “Obfuscated C Contest”, https://www.ioccc.org/
- Humans are different
  - Names should be distinguishable and meaningful
  - Comments should describe the intent and significant design decisions
  - Code should be structured in independent functions and modules
- Uniformity and readability matter

Naming, Naming, Naming

- variables: use reasonable and self-describing names, not too long
- index variables: i,j,k
- x,y are usually used for axes in a plot
- Structure your program, keep similar things together
- Use variables for values you need several times
- Read a style Guide

Comment your code
What does your code do?
What does it expect as input, which format?
Write your code for someone else
(you will be someone else in a few months ;))

Try not to swear or be inappropriate ;)

Always code as if the person who ends up maintaining your code is a violent psychopath who knows where you live.

Coding Style Examples

```
def main():
    a = int(input("Enter your age!\n"))
    if (a < 18):
        print(“You cannot legally drink in Finland!")
    else:
        print(“Enjoy your drink!")
```

```
def main():
    age = int(input("Enter your age!\n"))
    if (age < 18):
        print(“You cannot legally drink in Finland!")
    else:
        print(“Enjoy your drink!")
```

Coding Style Examples

```
def main():
    a = float(input("Enter number!\n"))
    b = float(input("Enter number!\n"))
    c = a*b
    print(“Your result is: “+c)
```

```
# This program calculates the product of two input factors
def main():
    factor1 = float(input("Enter your first factor!\n"))
    factor2 = float(input("Enter your second factor!\n"))
    product = factor1*factor2
    print(“Your product is: “+product)
```

Coding Style Examples

```
def main():
    aFloatNumberToCalculateTheProduct =float(input("Enter a floating point number!\n"))
    anotherFloatNumberToCalculateTheProduct = float(input("Enter a floating point number!\n"))
   variableToStoreTheProductOfTheNumbers = aFloatNumberToCalculateTheProduct*
anotherFloatNumberToCalculateTheProduct
    print(“Your product from your two floating point numbers is” + variableToStoreTheProductOfTheNumbers)
```

Coding Style Examples

```
def example1():
    nofYears = 10
    startCapital = 50
    interest = 0.01
    currentCapital = startCapital
    for i in range(nofyears):
        currentCapital += currentCapital*interest
    print(“after ” + nofYears + “ your starting capital of “+startCapital+” has become “+currentCapital)
```

```
def example2():
    while i<10:
        if i==0:
            money=50
        money += money*0.01
    print(“after 10 years your starting capital of 50 has become “+money)
```


"That's all Folks!"
Isberg