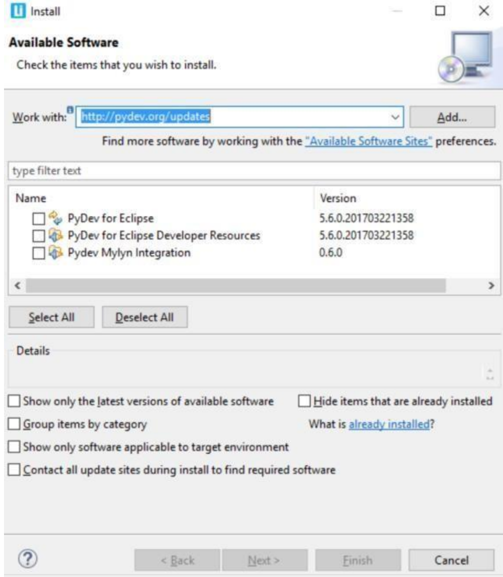
Introduction to Python

Lab 5

WHY PYTHON?

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive by using the prompt.**
- **It is object oriented.**



Install

Available Software

Check the items that you wish to install.

Work with: http://pydev.org/updates

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
□ PyDev for Eclipse	5.6.0.201703221358
□ PyDev for Eclipse Developer Resources	5.6.0.201703221358
□ Pydev Mylyn Integration	0.6.0

□ Show only the latest versions of available software

□ Hide items that are already installed

□ Group items by category

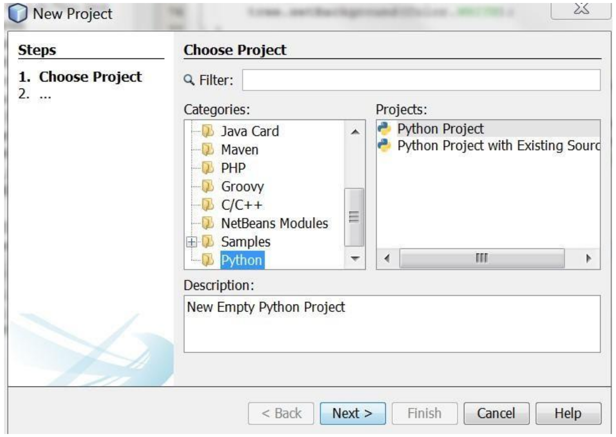
What is already installed?

□ Show only software applicable to target environment

□ Contact all update sites during install to find required software

INSTALLATION

- Installation
  - $ sudo apt-get install python3
  - **In the bash shell (Linux): type export PYTHONPATH=/usr/local/bin/python3.VersionX and press Enter.**
  - **For Eclipse PyDev needs to be installed. Go to Help > Install New Software...** Menu
  - **For Netbeans, the Python category will appear**



New Project

Steps

1. **Choose Project**
2. ...

Choose Project

Filter:

Categories:

- Java Card
- Maven
- PHP
- Groovy
- C/C++
- NetBeans Modules
- Samples
- Python

Projects:

- Python Project
- Python Project with Existing Sourc

Description:

New Empty Python Project

INTERACTIVE SHELL

- Type statements or expressions at prompt:
  - >>> print("Hello, world")
  - Hello, world
  - >>> x = 12**2
  - >>> x/2
  - 72
  - >>> # this is a comment

My first Python program

- Inside a .py program write the following lines of code:

```
__author___= “Student"
__date___= "$Nov 17, 2018 3:52:06 PM$"

if _name____== "__main__":
    print("Hello World")
```

BASIC TYPES: NUMBERS AND STRINGS

- Python supports four different numerical types: int, long, float, complex (a + b J, where J presents the square root of -1; a is the real part, b is the imaginary part).
- **Strings**
  - "hello"+"world“ => "helloworld"# concatenation
  - "hello"*3 => "hellohellohello" # repetition
  - “hello"[0] => "h“ # indexing
  - "hello"[-1] => "o“ # (from end)
  - "hello"[1:4] => "ell“ # slicing
  - len("hello") => 5 # size
  - "hello" < "jello“ => 1 # comparison
  - "e" in "hello“ => 1 # search

Container types: lists, dictionaries, tuples

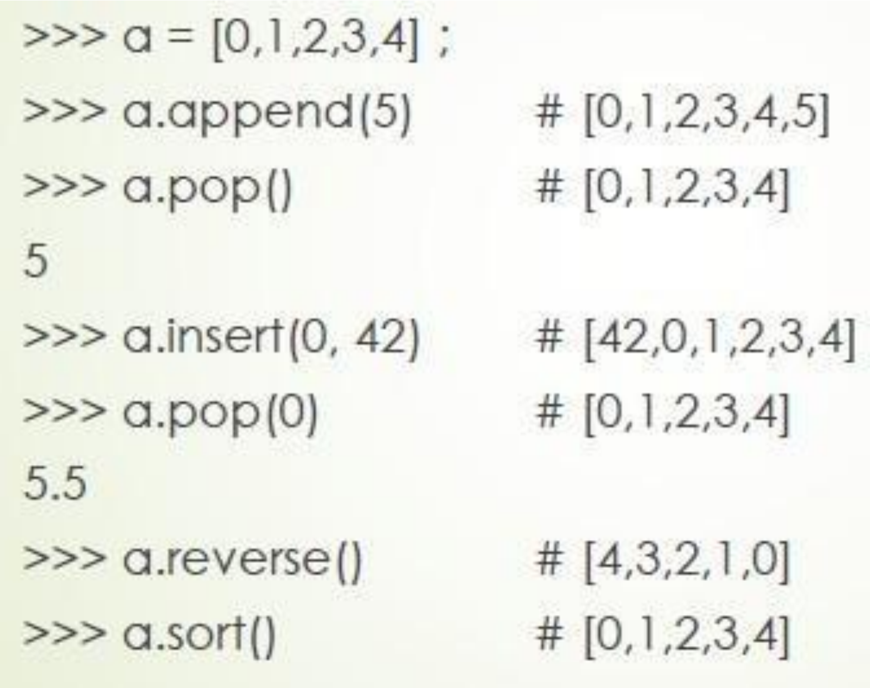
- **Lists**
  - a = [99, "bottles of beer", ["on", "the", "wall"]]
  - Same operators as for strings
  - The method append() appends a passed obj into the existing list.

Example

```
aList= [123, 'xyz', 'zara', 'abc'];
aList.append(2018);
print "Updated List : ", aList
```

When we run above program, it produces following result −Updated List : [123, 'xyz', 'zara', 'abc', 2018]

CONTAINER TYPE: LISTS



```
>>> a = [0,1,2,3,4] ;
>>> a.append(5)          # [0,1,2,3,4,5]
>>> a.pop()              # [0,1,2,3,4]
5
>>> a.insert(0, 42)      # [42,0,1,2,3,4]
>>> a.pop(0)             # [0,1,2,3,4]
5.5
>>> a.reverse()          # [4,3,2,1,0]
>>> a.sort()             # [0,1,2,3,4]
```

CONTAINER TYPE: DICTIONARIES

Hash tables, "associative arrays"

- d = {"duck": "eend", "water": "water"}
- Lookup:
  - d["duck"] -> "eend"
  - d["back"] # raises KeyError exception
- Delete, insert, overwrite:
  - del d["water"] # {"duck": "eend", "back": "rug"}
  - d["back"] = "rug" # {"duck": "eend", "back": "rug"}
  - d["duck"] = "duik" # {"duck": "duik", "back": "rug"}

CONTAINER TYPE: DICTIONARIES

Keys, values, items:

- d.keys() -> ["duck", "back"]
- d.values() -> ["duik", "rug"]
- d.items() -> [("duck","duik"), ("back","rug")]

Presence check:

- d.has_key("duck") -> 1; d.has_key("spam") -> 0

Values of any type; keys almost any

- {"name":"Guido", "age":43, ("hello","world"):1,42:"yes", "flag": ["red","white","blue"]}

CONTAINER TYPE: TUPLES

- A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Examples

  - tup1 =('physics','chemistry',1997,2000);
  - tup2 =(1,2,3,4,5);
  - tup3 ="a","b","c","d";

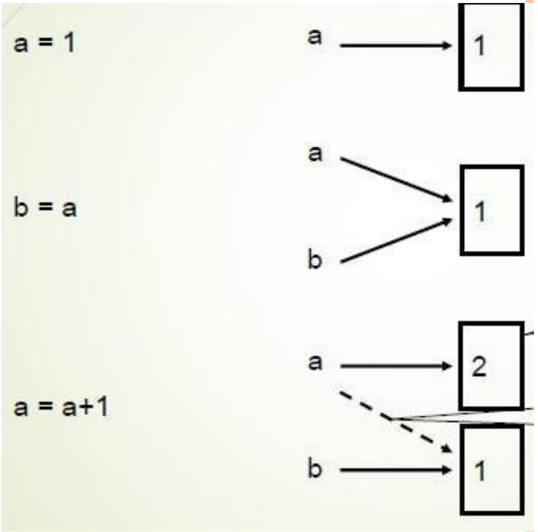
VARIABLES

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Not typed

  if friendly: greeting = "hello world"

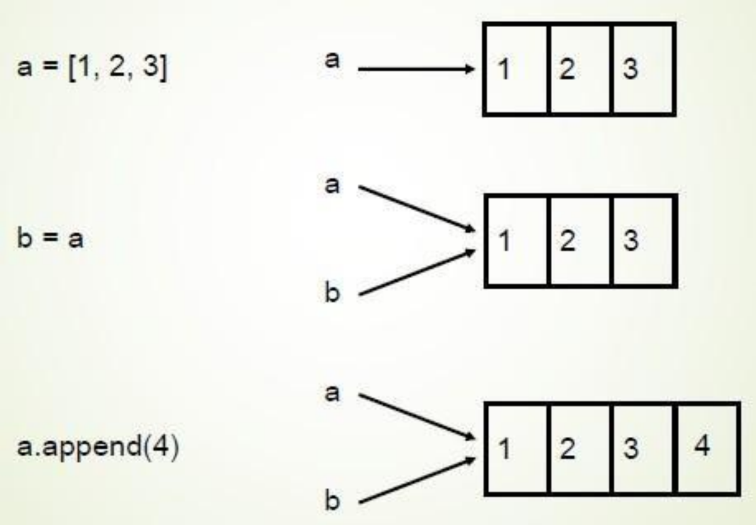
  else: greeting = 12**2

  print greeting

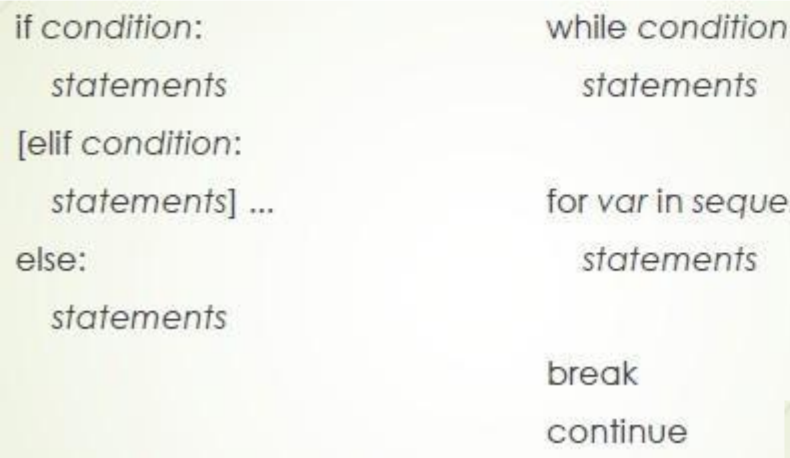


- Everything is a "variable":
  - Even functions, classes, modules

  Example:
  - >>> a = [1, 2, 3]
  - >>> b = a
  - >>> a.append(4)
  - >>> print b
  - [1, 2, 3, 4]



Control Structures

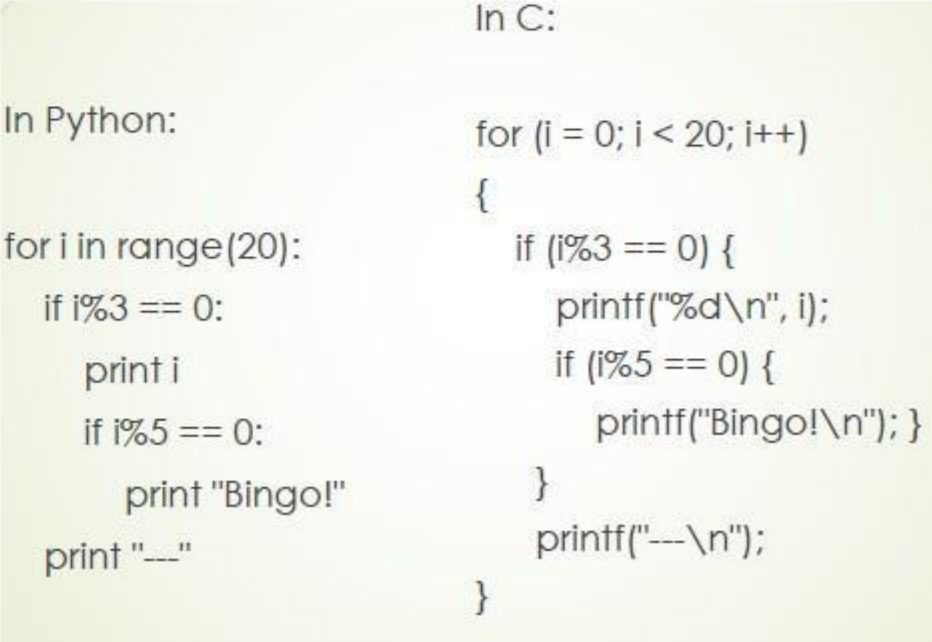


```
if condition:
    statements
[elif condition:
    statements] ...
else:
    statements
```

```
while condition:
    statements

for var in sequence:
    statements

break
continue
```



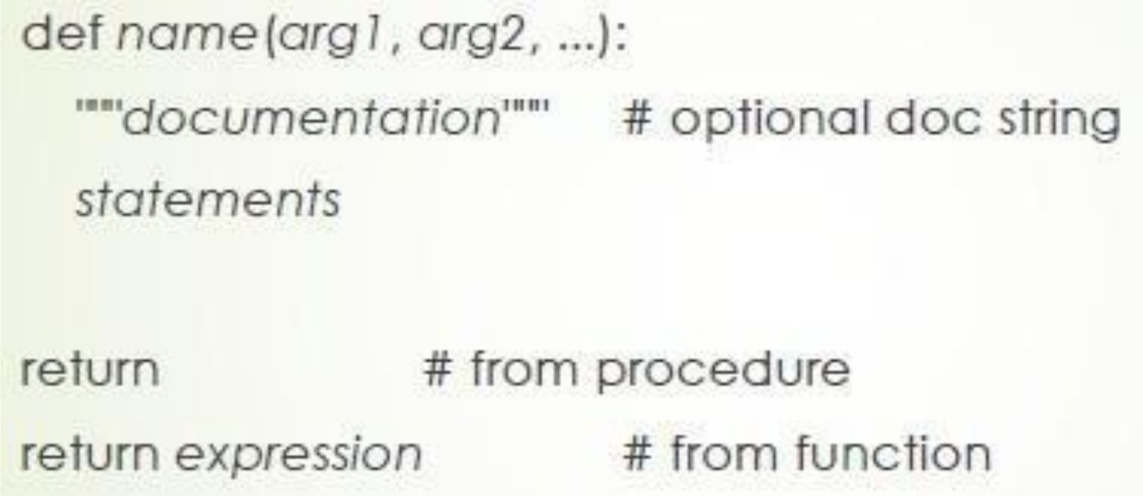
In Python:

```
for i in range(20):
    if i%3 == 0:
        print i
        if i%5 == 0:
            print "Bingo!"
    print "---"
```

In C:

```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n"); }
      }
      printf("---\n");
}
```

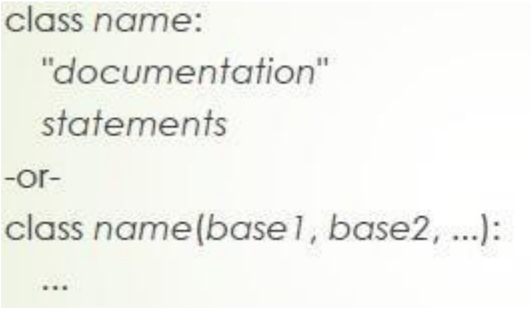
FUNCTIONS & PROCEDURES



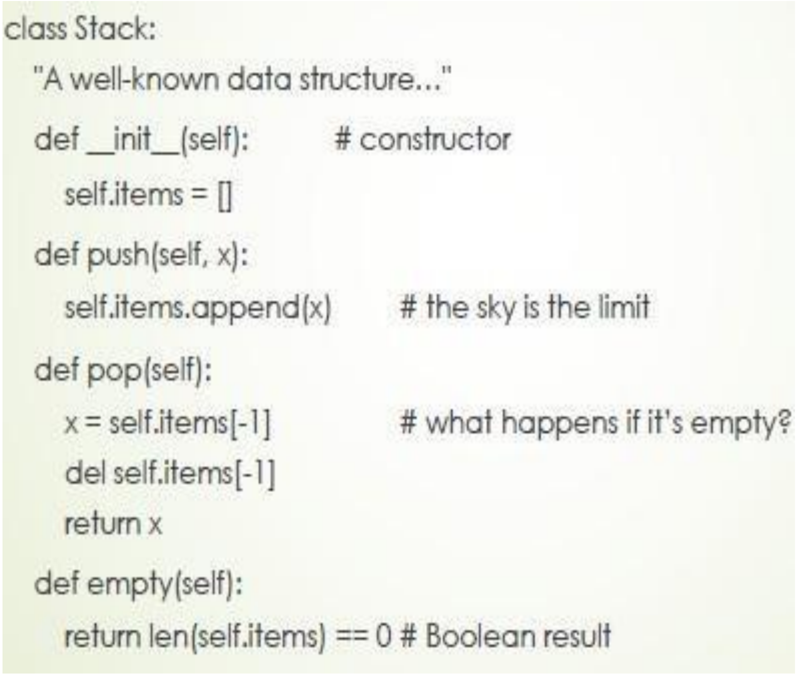
```
def name(arg1, arg2, ...):
    """documentation"""     # optional doc string
    statements


return                  # from procedure
return expression            # from function
```

CLASSES & INSTANCES



```
class name:
   "documentation"
   statements
-or-
class name(base1, base2, ...):
   ...
```



```
class Stack:
  "A well-known data structure..."
  def __init__(self):        # constructor
    self.items = []
  def push(self, x):
    self.items.append(x)      # the sky is the limit
  def pop(self):
    x = self.items[-1]        # what happens if it's empty?
    del self.items[-1]
    return x
  def empty(self):
    return len(self.items) == 0 # Boolean result
```

To create an instance, simply call the class object:

```
x = Stack()# no 'new' operator!
```

To use methods of the instance, call using dot notation:

```
x.empty() # -> 1
x.push(1) # [1]
x.empty() # -> 0
x.push("hello") # [1, "hello"]
x.pop() # -> "hello"# [1]
```

To inspect instance variables, use dot notation:

```
x.items# -> [1]
```

INSTANCES

- On use via instance (self.x), search order:
  - (1) instance, (2) class, (3) base classes
  - this also works for method lookup
- On assignment via instance (self.x = ...):
  - always makes an instance variable
- Class variables "default" for instance variables
- But...!
  - mutable *class variable: one copy shared by all*
  - mutable *instance variable: each instance its own*

MODULES & PACKAGES

- Modules
  - Collection of stuff in *foo.py file*
    - functions, classes, variables
  - Importing modules:
    - import re; print re.match("[a-z]+", s)
    - from re import match; print match("[a-z]+", s)
  - Import with rename:
    - import re as regex
- Packages
  - Collection of modules in directory
  - Must have__init__.py file
  - May contain subpackages
  - Import syntax:
    - from P.Q.M import foo; print foo()
    - from P.Q import M; print M.foo()
    - import P.Q.M; print P.Q.M.foo()
    - import P.Q.M as M; print M.foo()# new

EXCEPTIONS

- Catching exceptions

```
def foo(x):
    return 1/x
def bar(x):
    try:
        print foo(x)
    except ZeroDivisionError, message:
        print("Can’t divide by zero:", message)
bar(0)
```

FILES & STANDARD LIBRARY

- f = open(*filename[, mode[, buffersize])*
  - mode can be "r", "w", "a" (like C stdio); default "r"
  - append "b" for text translation mode
  - append "+" for read/write open
  - buffersize: 0=unbuffered; 1=line-buffered; buffered
- methods:
  - read([*nbytes]), readline(),readlines()*
  - write(*string), writelines(list)*
  - seek(*pos[, how]), tell()*
  - flush(), close()
  - fileno()

EXECISES

- Ex. 1. Write a program that greets a person, where the name of the person is read from the keyboard.
- Ex. 2. Write a program that reads four numbers (a, b, c, d) and computes the result of a +b * c + d.
- Hint:

```
if _name____== "__main__":
    a = input("a = ")
    b = input("b= ")
    multRes = int(a) * float(b)
    print(multRes)
```

EXERCICES

- Ex. 3. Consider the previous example, but this time you have to use methods and the numbers should have the values less than 10.

Hint:

```
if _name____== "__main__":
    def compute(a, b):
        if a > 10:
              Print(“a is too big")
        elif a <= 10:
               m = int(a) * int(b)
                print(“Mult: ", m)
        return “Result = " + str(m)

    a = input("Enter a: ");
    b = input("Enter b: ");
    print(compute(a, b))
```

EXERCISES

- Ex. 4. Consider the case when the user types a non-numeric input.

Hint:

```
import sys
 try:
        a = int(input("Enter a: "));
        b = float(input("Enter b: "));
 except:
            print("Error, please enter numeric input")
            sys.exit(1) #to end the program
```

Exercises

- Ex. 5. Write a program where the game Cows and Bulls is played and a user can quit when he/she types exit. At the end the total number of trials is shown and the average number of succes.

  Bulls: number of digits which are correctly placed

  1 2 3 4

  1 3 2 4

  2 bulls

  2 cows

- Ex. 6. Consider the case when you also want to know how much time that person spent until he/she did guess the number.
- Hint:

```
import time
start = time.time()
print("Let’s enjoy coding")
end = time.time()
print(end - start)
```

Exercises

- Ex. 7. Consider reading from a file and printing the whole text in uppercase in a separate file.