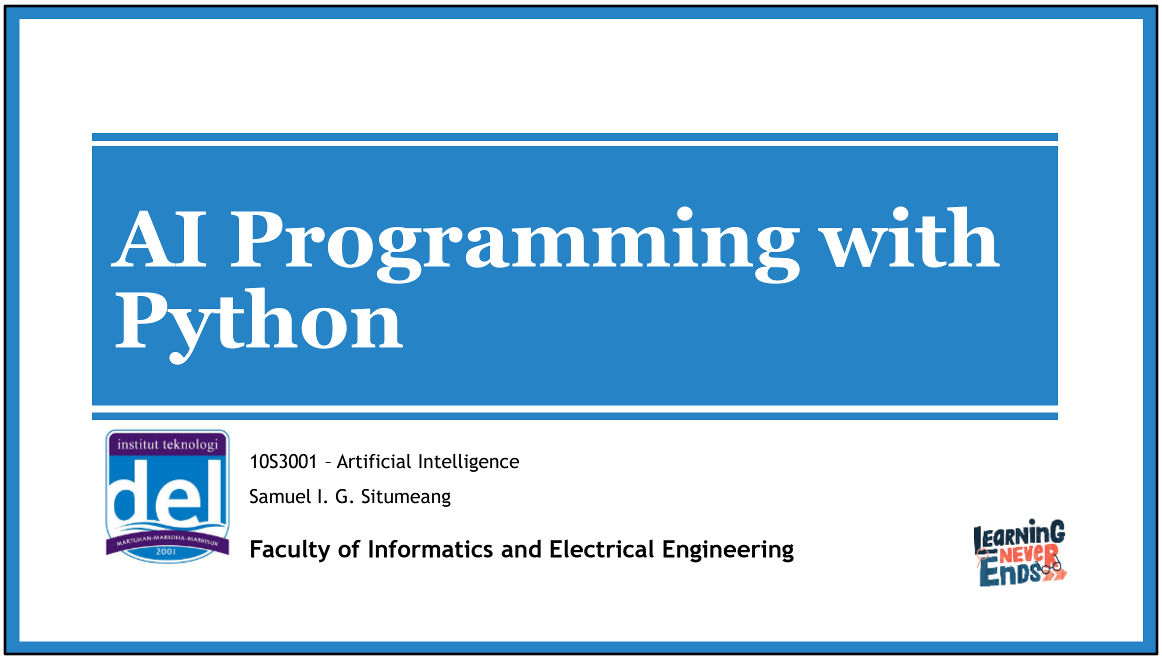
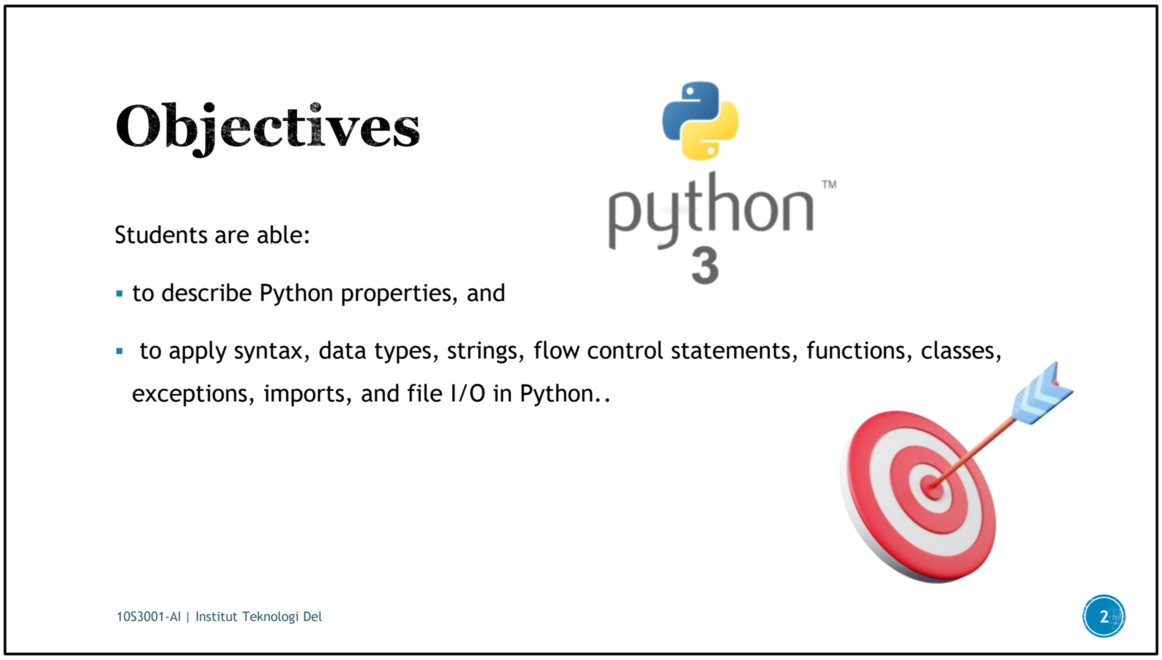


AI Programming with Python

10S3001 – Artificial Intelligence

Samuel I. G. Situmeang

Faculty of Informatics and Electrical Engineering



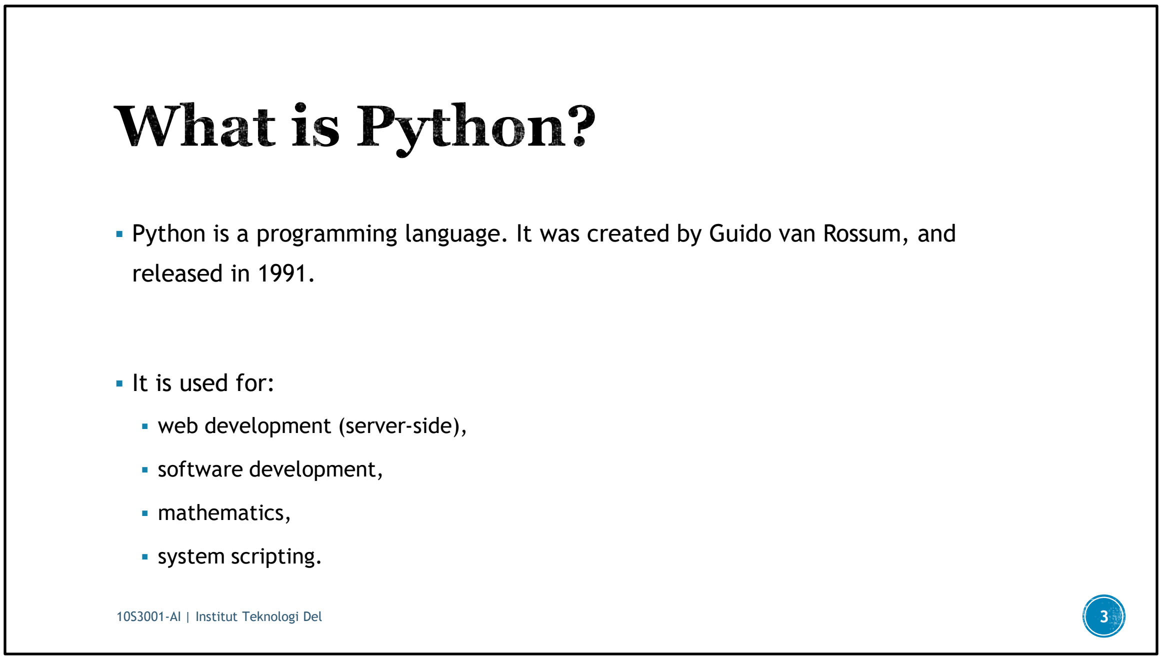
Objectives

Students are able:

- to describe Python properties, and
- to apply syntax, data types, strings, flow control statements, functions, classes, exceptions, imports, and file I/O in Python..

Siswa mampu:

- mendeskripsikan properti Python, dan
- menerapkan sintaksis, tipe data, string, pernyataan kontrol aliran, fungsi, kelas, pengecualian, impor, dan I/O file dengan Python..



What is Python?

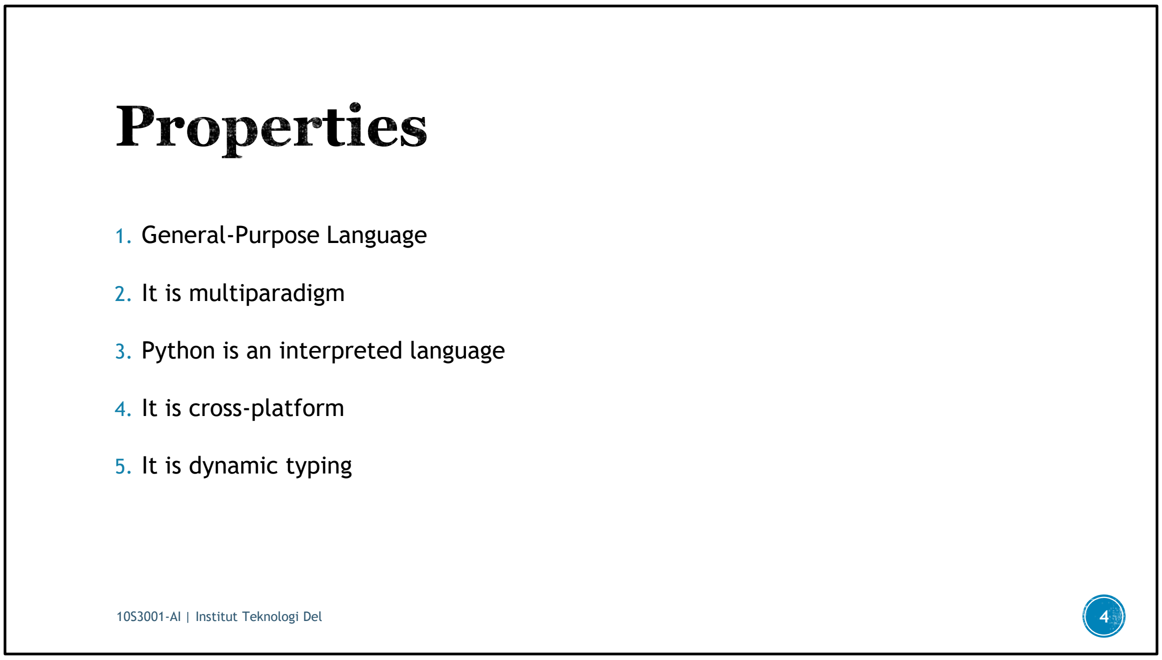
- Python is a programming language. It was created by Guido van Rossum, and released in 1991.

- It is used for:
  - web development (server-side),
  - software development,
  - mathematics,
  - system scripting.

10S3001-AI | Institut Teknologi Del

3

Python adalah bahasa pemrograman yang dibuat oleh Guido van Rossum dan dirilis pada tahun 1991. Bahasa ini digunakan untuk pengembangan web (sisi server), pengembangan perangkat lunak, matematika, *scripting* sistem.



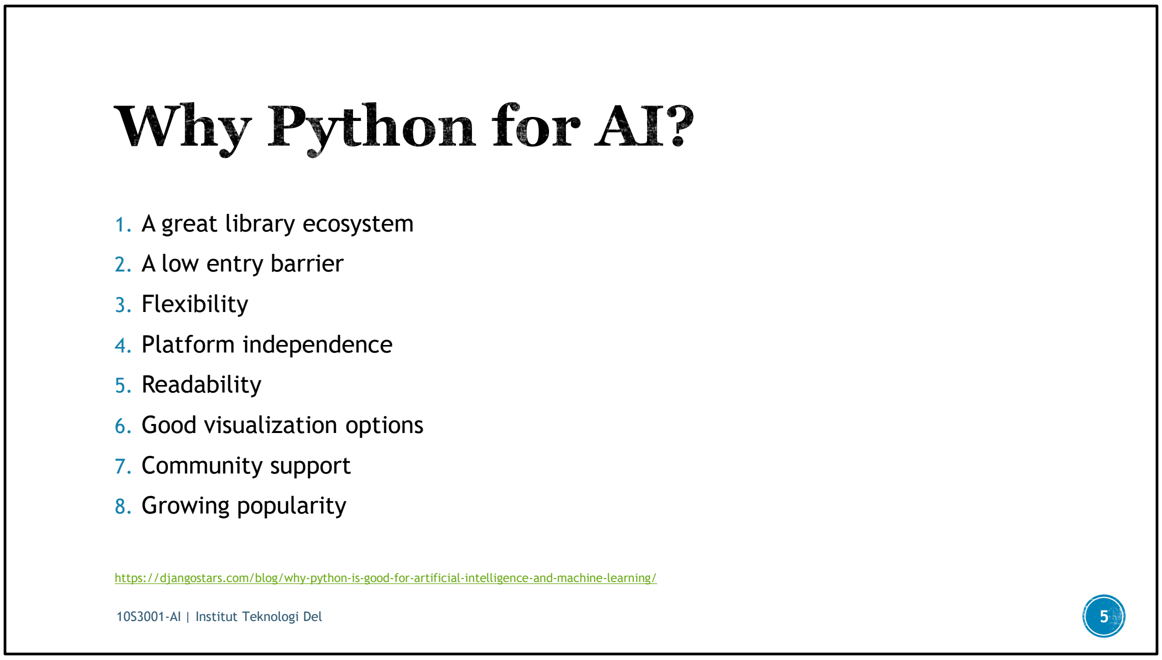
Properties

1. General-Purpose Language
2. It is multiparadigm
3. Python is an interpreted language
4. It is cross-platform
5. It is dynamic typing

10S3001-AI | Institut Teknologi Del

4

1. Python disebut *general-purpose language* karna dapat digunakan dalam berbagai aplikasi, seperti ilmu data, pengembangan perangkat lunak dan web, otomatisasi, dan penyelesaian pekerjaan secara umum.
2. Python adalah bahasa pemrograman multi-paradigma. Artinya mendukung berbagai gaya penulisan kode program. Kita dapat menulis kode program Python dengan paradigma prosedural, berorientasi objek, ataupun fungsional.
3. Python disebut sebagai bahasa ditafsirkan (*interpreted*). Secara teknis, Python dikompilasi menjadi *bytecode* dan kemudian diinterpretasikan dalam mesin virtual. Tidak ada langkah kompilasi eksplisit dalam Python seperti pada Java atau C. Dari sudut pandang pengguna, seakan-akan Python hanya menafsirkan file **.py** secara langsung. Selain itu, Python menawarkan *prompt* interaktif di mana kita dapat mengetikkan pernyataan-pernyataan Python dan segera mengeksekusinya. Jadi, alur kerja di Python jauh lebih mirip dengan bahasa yang ditafsirkan (*interpreted*) dibandingkan dengan bahasa yang dikompilasi (*compiled*).
4. Python bersifat lintas *platform*. Anda dapat menjalankan hampir semua program Python di Windows, Mac, perangkat keras Linux, dan bahkan Android dan iOS.



Why Python for AI?

1. A great library ecosystem
2. A low entry barrier
3. Flexibility
4. Platform independence
5. Readability
6. Good visualization options
7. Community support
8. Growing popularity

https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/

10S3001-AI | Institut Teknologi Del

5

Pilihan pustaka yang banyak adalah salah satu alasan utama Python menjadi bahasa pemrograman paling populer yang digunakan untuk AI. Pustaka adalah modul atau sekelompok modul yang diterbitkan oleh berbagai sumber seperti PyPi yang menyertakan potongan kode yang telah ditulis sebelumnya yang memungkinkan pengguna mengakses dan menggunaan beberapa fungsi. Contoh pustaka Python antara lain:

- Scikit-learn untuk menangani algoritma ML dasar seperti *clustering*, regresi linier dan logistik, regresi, klasifikasi, dan lain-lain.
- Pandas untuk struktur dan analisis data tingkat tinggi. Ini memungkinkan penggabungan dan pemfilteran data, serta pengumpulannya dari sumber eksternal lain seperti Excel, misalnya.
- Keras untuk pembelajaran mendalam (*deep learning*). Ini memungkinkan penghitungan dan pembuatan prototipe dengan cepat, karena menggunakan GPU selain CPU komputer.
- TensorFlow untuk bekerja dengan pembelajaran mendalam dengan menyiapkan, melatih, dan memanfaatkan jaringan saraf tiruan dengan kumpulan data yang sangat besar.
- Matplotlib untuk membuat plot 2D, histogram, bagan, dan bentuk visualisasi lainnya.

- NLTK untuk bekerja dengan linguistik komputasi, pengenalan dan pemrosesan bahasa alami.
- Scikit-image untuk pemrosesan gambar.
- PyBrain untuk jaringan saraf, pembelajaran tanpa pengawasan dan penguatan.
- Caffe untuk pembelajaran mendalam yang memungkinkan peralihan antara CPU dan GPU dan memproses 60+ juta gambar sehari menggunakan satu GPU NVIDIA K40.
- StatsModels untuk algoritma statistik dan eksplorasi data.

Di repositori PyPI, Anda dapat menemukan dan membandingkan lebih banyak pustaka Python.

Bahasa pemrograman Python menyerupai bahasa Inggris sehari-hari, sehingga proses belajarnya menjadi lebih mudah. Sintaksnya yang sederhana memungkinkan kita bekerja dengan nyaman bahkan dengan sistem yang kompleks, memastikan hubungan yang jelas antar elemen sistem. Itu sebabnya kita menggunakan Python untuk AI, ini semua tentang kesederhanaan.

Fleksibilitas Python memungkinkan pengembang memilih paradigma pemrograman yang mereka sukai atau bahkan menggabungkan gaya ini untuk menyelesaikan berbagai jenis persoalan dengan cara yang paling efisien. Ini memberi pengembang kebebasan terbesar untuk mengembangkan aplikasi AI.

Python dapat berjalan di platform apa pun termasuk Windows, MacOS, Linux, Unix, dan lainnya. Untuk mentransfer proses dari satu platform ke platform lainnya, pengembang hanya perlu melakukan beberapa perubahan skala kecil dan memodifikasi beberapa baris kode untuk membuat bentuk kode yang dapat dieksekusi untuk platform yang dipilih. Pengembang dapat menggunakan pustaka seperti PyInstaller untuk menyiapkan kode mereka agar dapat dijalankan pada platform berbeda. Hal ini menghemat waktu dan uang untuk pengujian di berbagai platform dan menjadikan keseluruhan proses pengembangan aplikasi AI lebih sederhana dan nyaman.

Python sangat mudah dibaca sehingga setiap pengguna dapat memahami kode rekannya dan mengubah, menyalin, atau membagikannya. Tidak ada kebingungan, kesalahan, atau paradigma yang bertentangan, dan hal ini menghasilkan pertukaran algoritme, ide, dan alat yang lebih efisien antara profesional AI.

Kita telah menyebutkan bahwa Python menawarkan beragam pustaka, dan beberapa di antaranya merupakan alat visualisasi yang hebat. Bagi pengembang AI, penting untuk digarisbawahi bahwa dalam kecerdasan buatan, sangat penting untuk dapat merepresentasikan data dalam format yang dapat atau lebih mudah dibaca manusia. Pustaka seperti Matplotlib memungkinkan kita membuat bagan, histogram, dan plot untuk pemahaman data yang lebih baik, presentasi yang efektif, dan visualisasi. Antarmuka pemrograman aplikasi yang berbeda juga menyederhanakan proses visualisasi dan memudahkan pembuatan laporan yang jelas.

Banyak dokumentasi Python tersedia online serta di komunitas dan forum Python, tempat pemrogram dan pengembang AI mendiskusikan kesalahan, menyelesaikan persoalan, dan saling membantu.

Sebagai hasil dari keunggulan yang telah dibahas tersebut, Python menjadi semakin populer di kalangan pengembang AI. Menurut Indeks TIOBE, ada lebih dari 265 bahasa pemrograman yang tersedia. Bahasa pemrograman yang paling popular pada tahun 2023 adalah Python.

Python libraries for General AI

- AIMA - Python implementation of algorithms from Russell and Norvig's "Artificial Intelligence: A Modern Approach".
- pyDatalog - Logic Programming engine in Python.
- SimpleAI - Python implementation of many of the artificial intelligence algorithms described on the book "Artificial Intelligence, A Modern Approach". It focuses on providing an easy to use, well documented and tested library.
- EasyAI - Simple Python engine for two-players games with AI (Negamax, transposition tables, game solving).

https://wiki.python.org/moin/PythonForArtificialIntelligence

Python Get Started

- To check if you have Python installed on a **Windows PC**, search in the start bar for Python or run the following on the Command Line (cmd.exe):

```
C:\Users\Your Name>python --version
```

- To check if you have python installed on a **Linux or Mac**, then on linux open the command line or on Mac open the Terminal and type:

```
python --version
```

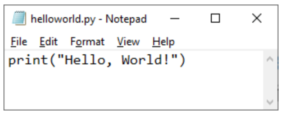
- If you find that you do not have python installed on your computer, then you can download it for free from the following website: https://www.python.org/ or you can install https://www.anaconda.com/.

Python Quickstart

- Python is an **interpreted programming language**.
  - Write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.
- The way to run a python file is like this on the command line:

```
C:\Users\Your Name>python helloworld.py
```

  - Where "**`helloworld.py`**" is the name of your python file.
- Let's write our first Python file, called **`helloworld.py`**, which can be done in any text editor.



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

Python Quickstart (*cont.*)

- Save your file. Open your command line, navigate to the directory where you saved your file, and run:

```
C:\Users\Your Name>python helloworld.py
```

- The output should read:

```
Hello, World!
```

The Python Command Line

- To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file.
- This is made possible because Python can be run as a command line itself.
- Type the following on the Windows, Mac or Linux command line:

```
python
```

- From there you can write any python, including our hello world example from earlier.

Getting help

- Call `help(<object>)` to know how an object works.
  - Integers, strings, floating point numbers, even arrays and dictionaries, are all **objects**.
  - More specifically, any single integer or any single string is an **object**.
  - The number 12 is an **object**, the string "hello, world!" is an object, a list is an object that can hold other objects, and so on.
- Call `dir(<object>)` to show you all the object's methods.
- Call `<object>.__doc__` to show you its documentation string.

```
>>> help(5)
Help on int object:

(etc etc)

>>> dir(5)
['__abs__', '__add__', ...]

>>> abs.__doc__
'abs(number) -> number

Return the absolute value of the argument.
```

Syntax

- Python has **no mandatory statement termination characters** and **blocks are specified by indentation**.
- Indent to begin a block, dedent to end one.
- Statements that expect an indentation level end in a colon (`:`).
- **Comments** start with the pound (`#`) sign and are single-line, multi-line strings are used for **multi-line comments**.
- **Values are assigned** (in fact, objects are **bound** to names) with the *equals sign* (“`=`”), and equality testing is done using two *equals signs* (“`==`”).
- You can increment/decrement values using the `+=` and `-=` operators respectively by the right-hand amount. This works on many datatypes, strings included.
- You can also use multiple variables on one line.

Syntax

```
>>> myvar = 3
>>> myvar += 2
>>> myvar
5
>>> myvar -= 1
>>> myvar
4
"""This is a multiline comment.
The following lines concatenate the two strings."""
>>> mystring = "Hello"
>>> mystring += " world."
>>> print(mystring)
Hello world.
# This swaps the variables in one line(!).
# It doesn't violate strong typing because values aren't
# actually being assigned, but new objects are bound to
# the old names.
>>> myvar, mystring = mystring, myvar
```

Data types

- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:
  - Text Type : str
  - Numeric Types : int, float, complex
  - Sequence Types : list, tuple, range
  - Mapping Type : dict
  - Set Types : set, frozenset
  - Boolean Type : bool
  - Binary Types : bytes, bytearray, memoryview

Data types

- Four collection data types:
  - **Set** is a collection which is unordered and unindexed. No duplicate members. Written with curly brackets.
  - **List** is a collection which is ordered and changeable. Allows duplicate members. Written with square brackets.
  - **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members. Written with curly brackets, and they have keys and values.
  - **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members. Written with round brackets.
- The index of the first item in all array types is 0.
- Negative numbers count from the end towards the beginning, -1 is the last item.
- Variables can point to functions.

Data types

```
>>> myset = {"cabbage", "onion", "tomato"}
>>> print(myset)
>>> sample = [1, ["another", "list"], ("a", "tuple")]
>>> mylist = ["List item 1", 2, 3.14]
>>> mylist[0] = "List item 1 again" # We're changing the item.
>>> mylist[-1] = 3.21 # Here, we refer to the last item.
>>> mydict = {"Key 1": "Value 1", 2: 3, "pi": 3.14}
>>> mydict["pi"] = 3.15 # This is how you change dictionary values.
>>> mytuple = (1, 2, 3)
>>> myfunction = len
>>> print(myfunction(mylist))
3
```

Data types

- You can access **array ranges** using a colon (`:`).
  - Leaving the start index empty assumes the first item, leaving the end index assumes the last item.
- Indexing is inclusive-exclusive, so specifying `[2:10]` will return items `[2]` (the third item, because of 0-indexing) to [9] (the tenth item), inclusive (8 items).
- Negative indexes count from the last item backwards (thus -1 is the last item).

Data types

```
>>> mylist = ["List item 1", 5, 6.31]
>>> print(mylist[:])
['List item 1', 5, 6.31]
>>> print(mylist[0:2])
['List item 1', 5]
>>> print(mylist[-3:-1])
['List item 1', 5]
>>> print(mylist[1:])
[5, 6.31]
# Adding a third parameter, "step" will have Python step in
# N item increments, rather than 1.
# E.g., this will return the first item, then go to the third and
# return that (so, items 0 and 2 in 0-indexing).
>>> print(mylist[::2])
['List item 1', 6.31]
```

Strings

- Strings can use either single or double quotation marks, and you can have quotation marks of one kind inside a string that uses the other kind (i.e. `"He said 'hello'."`).
- Multi-line strings are enclosed in *triple double (or single) quotes* (`"""`).
- To fill a string with values, you use the `%` (modulo) operator and a tuple.
- Each %s gets replaced with an item from the tuple, left to right, and you can also use dictionary substitutions.

Strings

```
>>> print("Name: %s\
Number: %s\
String: %s" % ("Budi", 9, 9 * "-"))
Name: Budi
Number: 9
String: ---

>>> strString = """This is
a multiline
string."""
print(strString)
This is
a multiline
string.

# WARNING: Watch out for the trailing s in "%(key)s".
>>> print("This %(verb)s a %(noun)s." % {"noun": "test", "verb": "is"})
This is a test.

>>> name = "Budi"
>>> "Hello, {}!".format(name)
'Hello, Budi!'
>>> print(f"Hello, {name}!")
Hello, Budi!
```

Flow control statements

- Flow control statements are `if`, `for`, and `while`.
- There is no `switch`; instead, use `if`.
- Use `for` to enumerate through members of a list.
- To obtain a sequence of numbers you can iterate over, use `range(<number>)`.

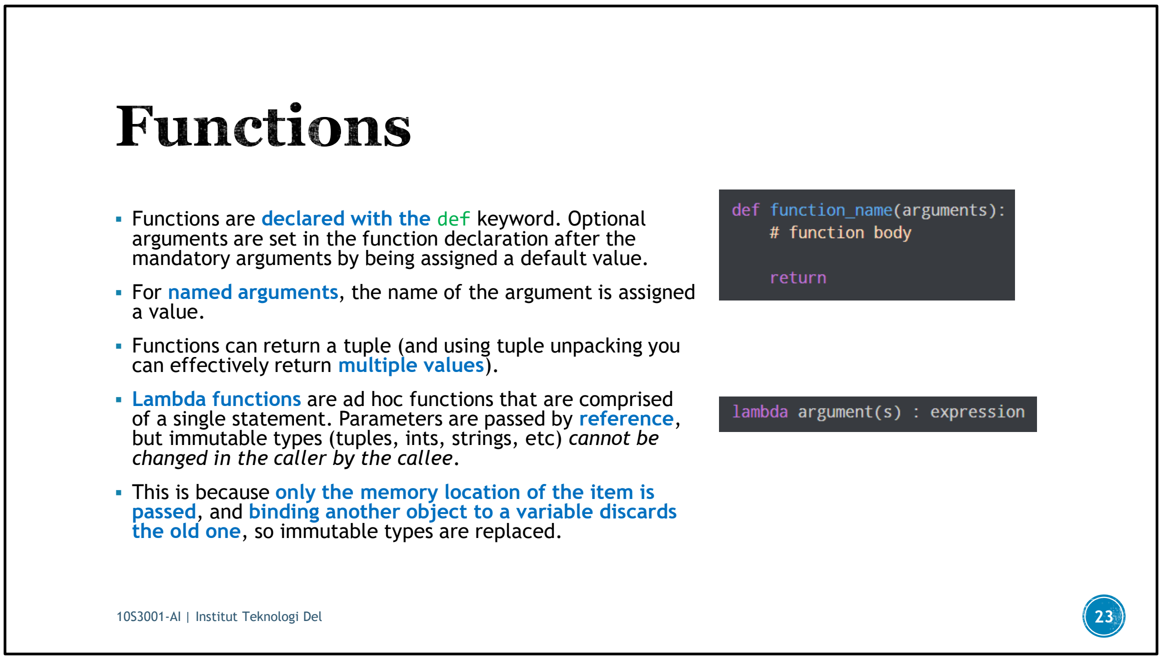
Flow control statements

```
>>> print(range(10))
range(0, 10)
>>> rangelist = list(range(10))
>>> print(rangelist)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> for number in range(10):
...     # Check if number is one of
...     # the numbers in the tuple.
...     if number in (3, 4, 7, 9):
...         # "Break" terminates a for without
...         # executing the "else" clause.
...         break
...     else:
...         # "Continue" starts the next iteration
...         # of the loop. It's rather useless here,
...         # as it's the last statement of the loop.
...         continue
...else:
...     # The "else" clause is optional and is
...     # executed only if the loop didn't "break".
...     pass # Do nothing

>>> if rangelist[1] == 2:
...     print("The second item (lists are 0-based) is 2")
...elif rangelist[1] == 3:
...     print("The second item (lists are 0-based) is 3")
...else:
...     print("Dunno")

>>> while rangelist[1] == 1:
...     print("We are trapped in an infinite loop!")
```



Functions

- Functions are **declared with the def** keyword. Optional arguments are set in the function declaration after the mandatory arguments by being assigned a default value.
- For **named arguments**, the name of the argument is assigned a value.
- Functions can return a tuple (and using tuple unpacking you can effectively return **multiple values**).
- **Lambda functions** are ad hoc functions that are comprised of a single statement. Parameters are passed by **reference**, but immutable types (tuples, ints, strings, etc) *cannot be changed in the caller by the callee*.
- This is because **only the memory location of the item is passed**, and **binding another object to a variable discards the old one**, so immutable types are replaced.

```
def function_name(arguments):
    # function body

    return
```

```
lambda argument(s) : expression
```

10S3001-AI | Institut Teknologi Del

23

Fungsi dideklarasikan dengan kata kunci **def**. Argumen opsional ditetapkan dalam deklarasi fungsi setelah argumen wajib dengan diberi nilai *default*. Contoh:

```
def function(a, b=999):
  return a+b

print(function(2, 3))
```

Perhatikan bahwa argumen kedua (b) pada fungsi tersebut bersifat opsional. Dalam kata lain, apabila dalam pemanggilan kita tidak *assign* nilai apapun, misalnya print(function(2)), maka fungsi akan menggunakan nilai 999 sebagai nilai untuk argumen kedua (b)

Untuk *named argument*, nama argumen tersebut diberi nilai. Contoh:

```
print(function(2, b=10))
```

Dalam hal ini, b kita sebut *named argument*.

Fungsi dapat mengembalikan *tuple* (dan menggunakan *tuple unpacking* Anda dapat mengembalikan beberapa nilai secara efektif). Contoh:

```
def function(a, b=999):
  c=a+b
  return (a, b, c)

(a, b, c) = function(2, b=10)
print(c)
```

Fungsi Lambda adalah fungsi ad hoc yang terdiri dari satu pernyataan. Parameter diteruskan dengan referensi, tetapi tipe data yang tidak dapat diubah (tuple, int, string, dll) tidak dapat diubah di *caller* oleh *callee*.

Hal ini karena hanya lokasi memori item yang diteruskan, dan mengikat objek lain ke variabel akan membuang objek lama, sehingga tipe yang tidak dapat diubah akan diganti.

Functions

```
# Same as def funcvar(x): return x + 1
>>> funcvar = lambda x: x + 1
>>> print(funcvar(1))
2

# an_int and a_string are optional, they have default values
# if one is not passed (2 and "A default string", respectively).
>>> def passing_example(a_list, an_int=2, a_string="A default string"):
...     a_list.append("A new item")
...     an_int = 4
...     return a_list, an_int, a_string

>>> my_list = [1, 2, 3]
>>> my_int = 10
>>> print(passing_example(my_list, my_int))
([1, 2, 3, 'A new item'], 4, "A default string")
>>> my_list
[1, 2, 3, 'A new item']
>>> my_int
10
```

Classes

- A Class is like an **object constructor**, or a "blueprint" for creating objects.
- Create a class named `MyClass`, with a property named `x`:

```
class MyClass:
  x = 5
```

- Use `MyClass` to create an object named `p1`, and print the value of `x`:

```
p1 = MyClass()
print(p1.x)
```

Classes

- All classes have a function called `__init__()`, which is always executed when the class is being initiated.
- Use the `__init__()` function to assign values to object properties, or other operations that are necessary to do when the object is being created.
- Create a class named Person, use the `__init__()` function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person(“Budi", 20)

print(p1.name)
print(p1.age)
```

Classes

- Objects can also contain methods. Methods in objects are functions that belong to the object.
- Let us create a method in the Person class.
  - Insert a function that prints a greeting, and execute it on the p1 object.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person(“Budi", 20)
p1.myfunc()
```

Classes

- The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
  - It does not have to be named `self`, you can call it whatever you like, but it has to be the first parameter of any function in the class.
- Use the words *myobject* and *abc* instead of *self*:

```
class Person:
  def __init__(myobject, name, age):
    myobject.name = name
    myobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person(“Budi", 20)
p1.myfunc()
```

Classes

- Modify properties on objects like this.
  - Set the age of p1 to 10.

```
p1.age = 40
```

- Delete properties on objects by using the del keyword.
  - Delete the age property from the p1 object.

```
del p1.age
```

- Delete objects by using the del keyword.
  - Delete the p1 object.

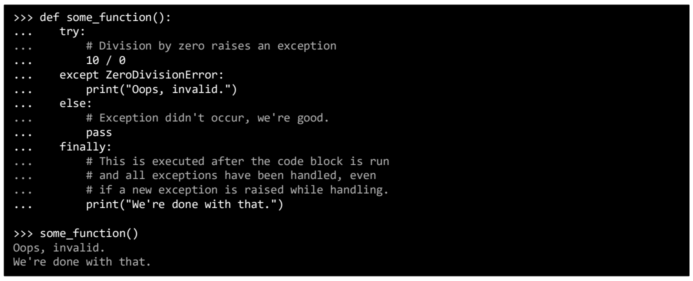
```
del p1
```

- Class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

```
class Person:
  pass
```

Exceptions

- Exceptions in Python are handled with **try-except [exceptionname]** blocks:



```
>>> def some_function():
...     try:
...         # Division by zero raises an exception
...         10 / 0
...     except ZeroDivisionError:
...         print("Oops, invalid.")
...     else:
...         # Exception didn't occur, we're good.
...         pass
...     finally:
...         # This is executed after the code block is run
...         # and all exceptions have been handled, even
...         # if a new exception is raised while handling.
...         print("We're done with that.")

>>> some_function()
Oops, invalid.
We're done with that.
```

Importing

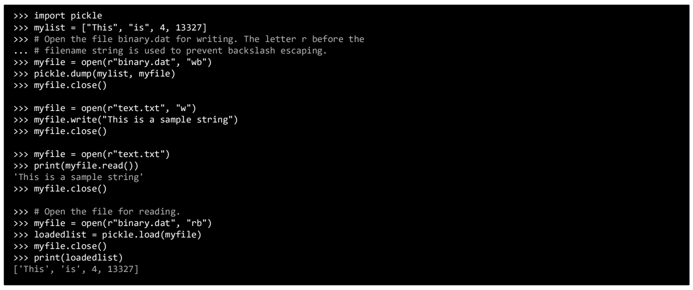
- External libraries are used with the `import [libname]` keyword.
- You can also use `from [libname] import [funcname]` for individual functions.

```
>>> import random
>>> from time import clock

>>> randomint = random.randint(1, 100)
>>> print(randomint)
64
```

File I/O

- Python has a wide array of libraries built in.
- As an example, here is how serializing (converting data structures to strings using the **pickle** library) with file I/O is used.



```
>>> import pickle
>>> mylist = ["This", "is", 4, 13327]
>>> # Open the file binary.dat for writing. The letter r before the
... # filename string is used to prevent backslash escaping.
>>> myfile = open(r"binary.dat", "wb")
>>> pickle.dump(mylist, myfile)
>>> myfile.close()

>>> myfile = open(r"text.txt", "w")
>>> myfile.write("This is a sample string")
>>> myfile.close()

>>> myfile = open(r"text.txt")
>>> print(myfile.read())
'This is a sample string'
>>> myfile.close()

>>> # Open the file for reading.
>>> myfile = open(r"binary.dat", "rb")
>>> loadedlist = pickle.load(myfile)
>>> myfile.close()
>>> print(loadedlist)
['This', 'is', 4, 13327]
```

Miscellaneous

- **Conditions can be chained**: `1 < a < 3` checks that a is both less than 3 and greater than 1.
- You can use `del` to **delete variables or items in arrays**.
- **List comprehensions** provide a powerful way to create and manipulate lists.
  - They consist of an expression followed by a `for` clause followed by zero or more `if` or `for` clauses.

Miscellaneous

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [3, 4, 5]
>>> print([x * y for x in lst1 for y in lst2])
[3, 4, 5, 6, 8, 10, 9, 12, 15]
>>> print([x for x in lst1 if 4 > x > 1])
[2, 3]
# Check if a condition is true for any items.
# "any" returns true if any item in the list is true.
>>> any([i % 3 for i in [3, 3, 4, 4, 3]])
True
# This is because 4 % 3 = 1, and 1 is true, so any()
# returns True.

# Check for how many items a condition is true.
>>> sum(1 for i in [3, 3, 4, 4, 3] if i == 4)
2
>>> del lst1[0]
>>> print(lst1)
[2, 3]
>>> del lst1
```

Miscellaneous

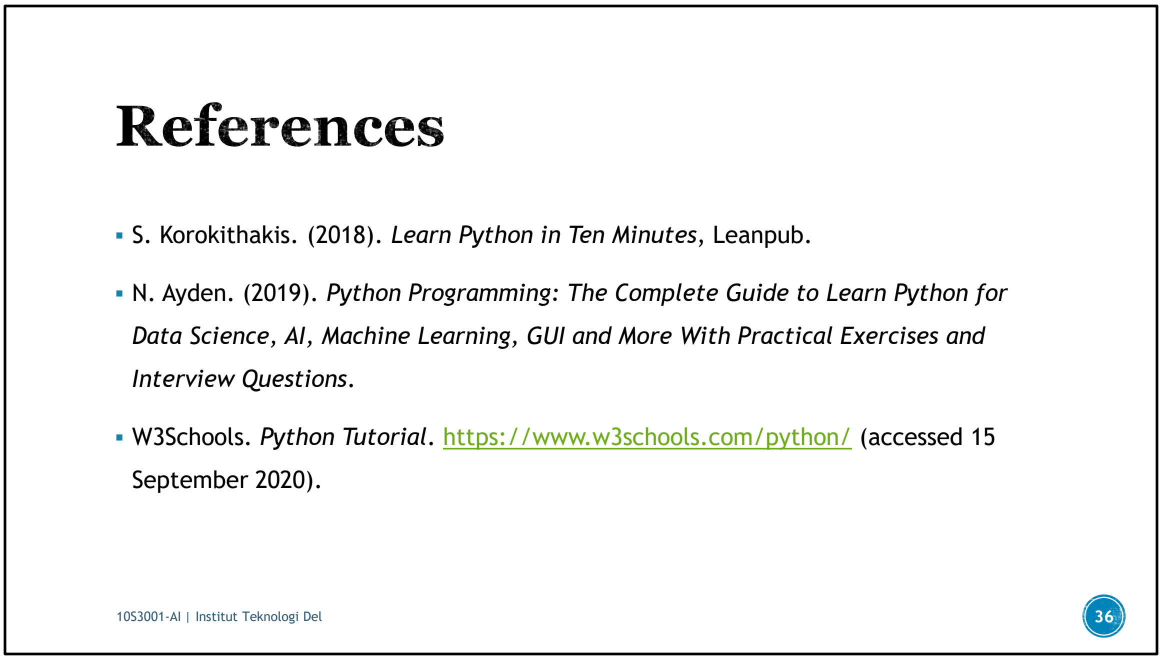
- **Global variables** are declared outside of functions and can be read without any special declarations.
  - But if you want to write to them, you must declare them at the beginning of the function with the `global` keyword, otherwise Python will bind that object to a new local variable (be careful of that, it's a small catch that can get you if you don't know it).

```
number = 5

def myfunc():
    # This will print 5.
    print(number)

def anotherfunc():
    # This raises an exception because the variable has not
    # been bound before printing. Python knows that it an
    # object will be bound to it later and creates a new, local
    # object instead of accessing the global one.
    print(number)
    number = 3

def yetanotherfunc():
    global number
    # This will correctly change the global.
    number = 3
```



References

- S. Korokithakis. (2018). *Learn Python in Ten Minutes*, Leanpub.
- N. Ayden. (2019). *Python Programming: The Complete Guide to Learn Python for Data Science, AI, Machine Learning, GUI and More With Practical Exercises and Interview Questions.*
- W3Schools. *Python Tutorial*. https://www.w3schools.com/python/ (accessed 15 September 2020).

Further Reading

- Python Software Foundation. *The Python Tutorial*. https://docs.python.org/3/tutorial/index.html.
- GitHub. *AIMA Python*. https://github.com/aimacode/aima-python.

eof