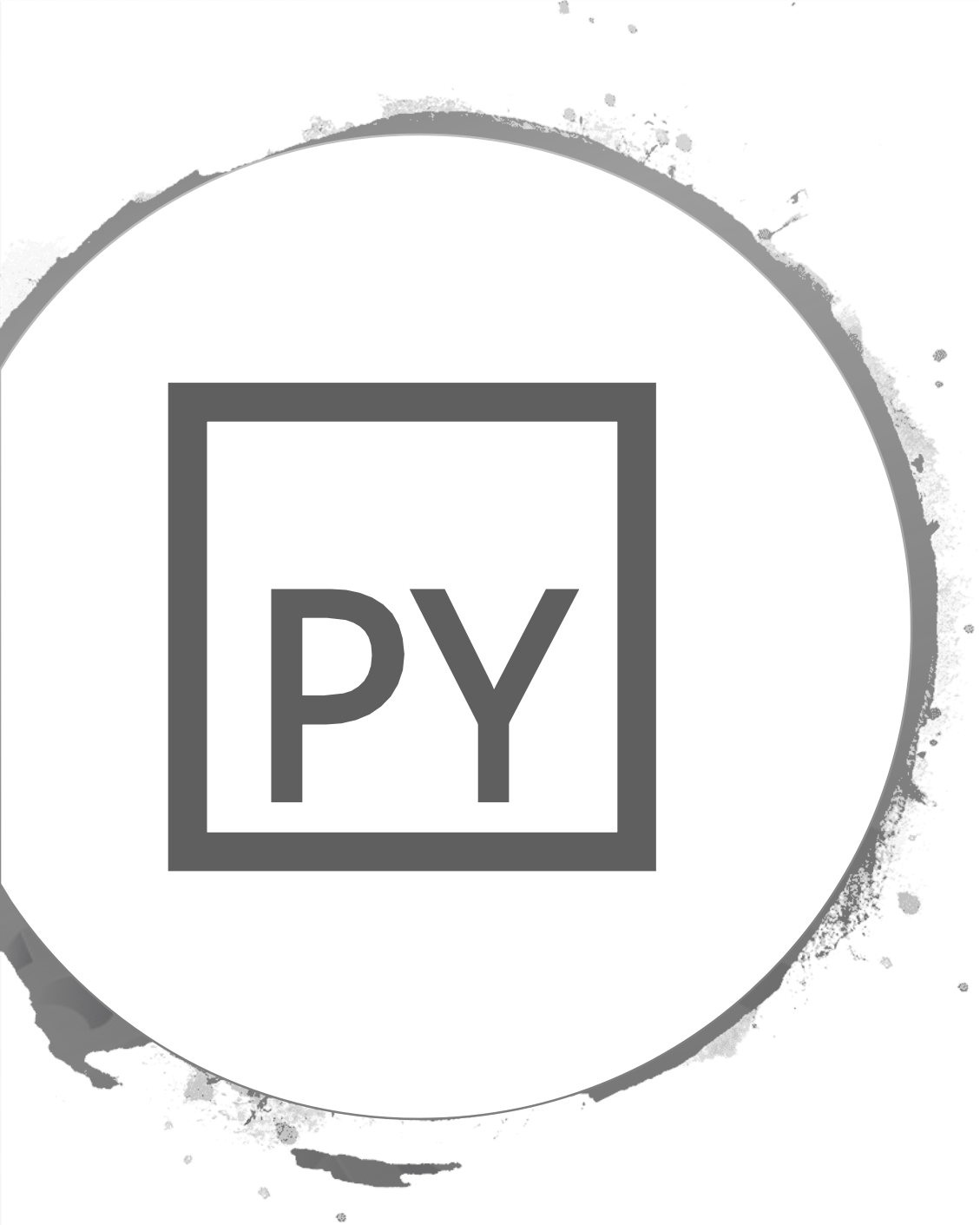


Programming in Python

AI Workshop, 2-4 December 2019

Somnuk Phon-Amnuaisuk
School of Computing and Informatics
Centre for Innovative Engineering
Universiti Teknologi Brunei


meme
A meme (/ˈ mi: m/ meem), a neologism coined by Richard Dawkins, is "an idea, behavior, or style that spreads from person to person within a culture". A meme acts as a unit for carrying cultural ideas, symbols, or practices that can be transmitted from one mind to another through writing, speech, gestures, rituals, or other imitable phenomena with a mimicked theme.

Disclaimer

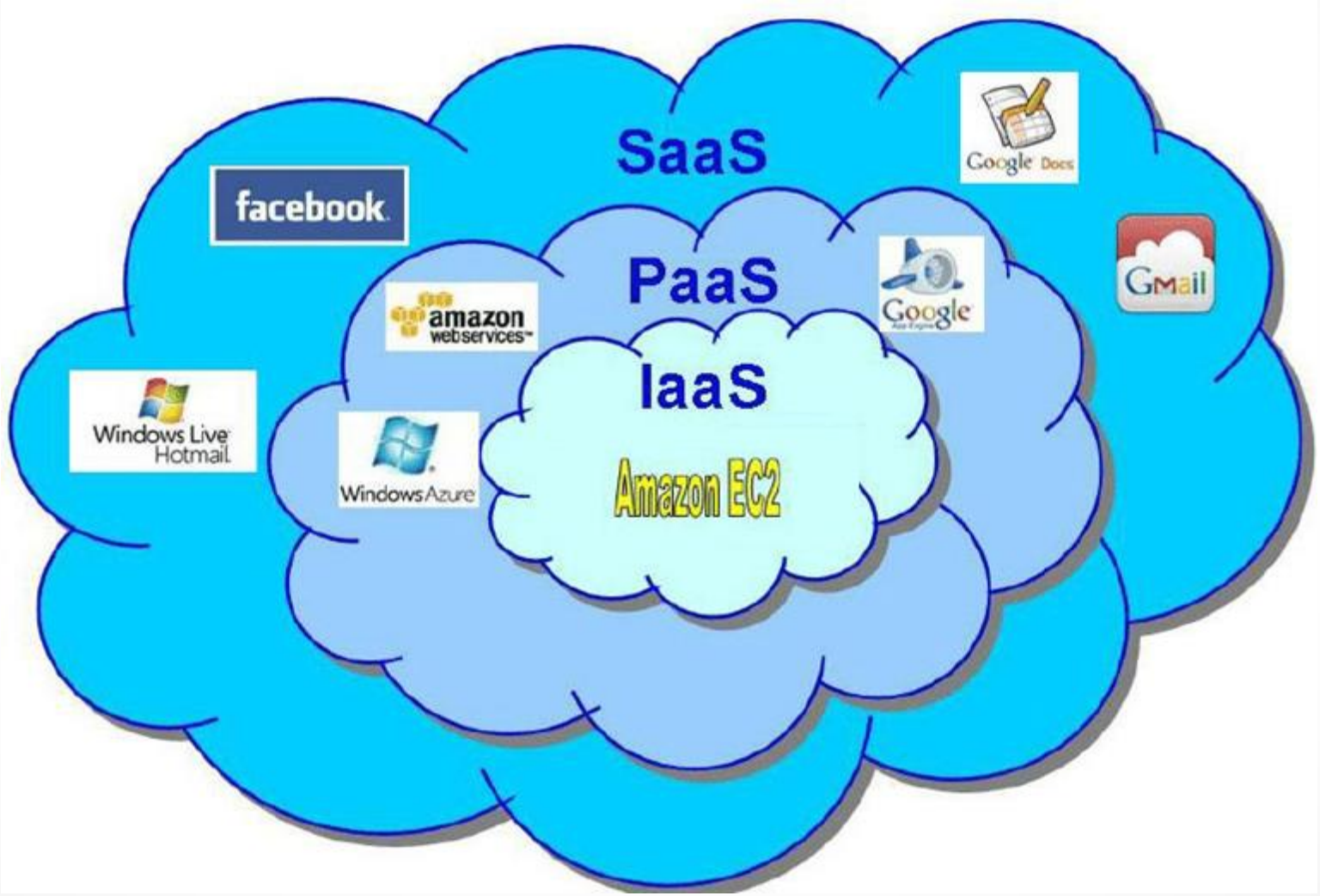
This lecture is compiled from my lectures as well as materials gathering from other lectures found in the public domains.

Learning Outcomes

- Able to use Google Colaboratory cloud app services
- Able to describe various Python data types
- Able to describe and use various Python modules
- Able to use Python as a programmable calculator
- Able to define and create functions, class and modules
- Able to create a Python program to solve a simple problem

Outline

- Cloud Application Services
  - Introduction to Google Colaboratory
  - How to code and comments in Jupyter notebooks
- Python Programming
  - A brief historical facts
  - Syntax, data and operations
  - Functions, classes and modules
- Exercises


SaaS
PaaS
IaaS
facebook
Google Docs
amazon web services
Google
Gmail
Windows Live Hotmail
Windows Azure
Amazon EC2

Cloud Application Services

What is colaboratory?

- Colaboratory is a cloud application service from Google.
- Colab is an excellent research tool for machine learning education and research.
- Colab provides a Jupyter notebook environment that requires no setup to use.
- Colab works with most major browsers, and is most thoroughly tested with latest versions of Chrome, Firefox and Safari.
- It is free ☺

What is colaboratory?

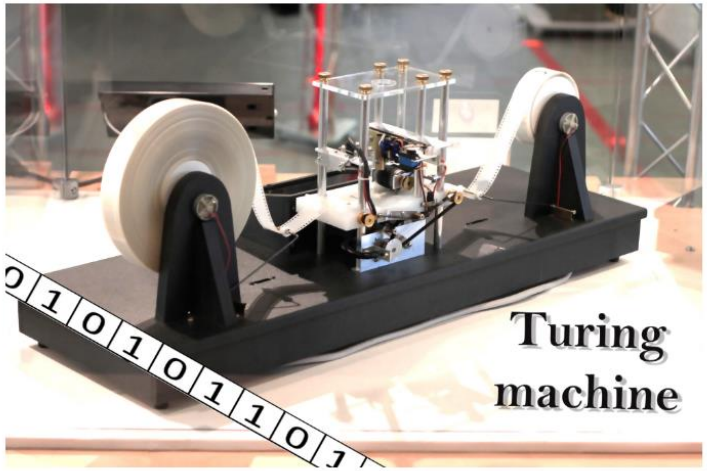
- Colaboratory is a cloud application service from Google.
- Colab is an excellent research tool for machine learning education and research.
- Colab provides a Jupyter notebook environment that requires no setup to use.
- Colab works with most major browsers, and is most thoroughly tested with latest versions of Chrome, Firefox and Safari.
- It is free ☺
- Let’s start, open Chrome and search for ‘Google Colab’

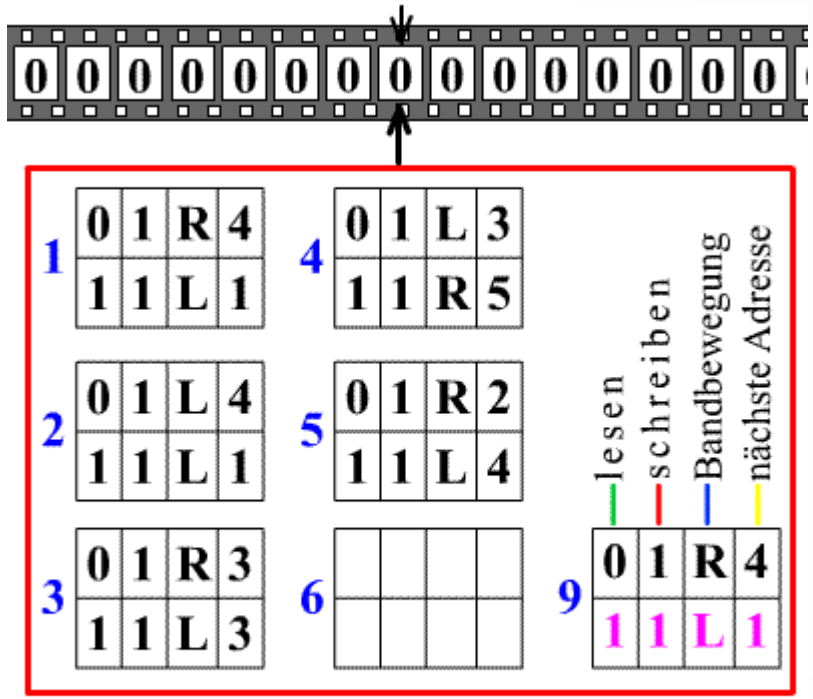
Python Programming

What is Python?

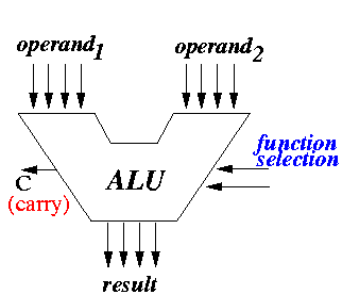
- Python is an interpreted, high-level, general-purpose programming language.
- Python is created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.
- There are two major versions python 2.x and python 3.x

What is Computing?

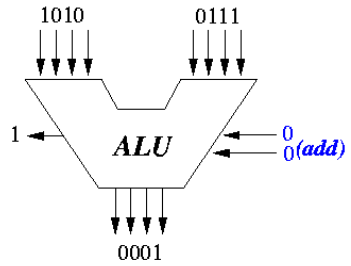




What is Computing?



Example operation:

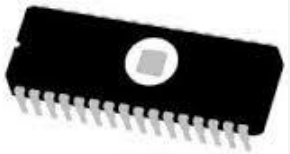


The ALU is performing this addition:

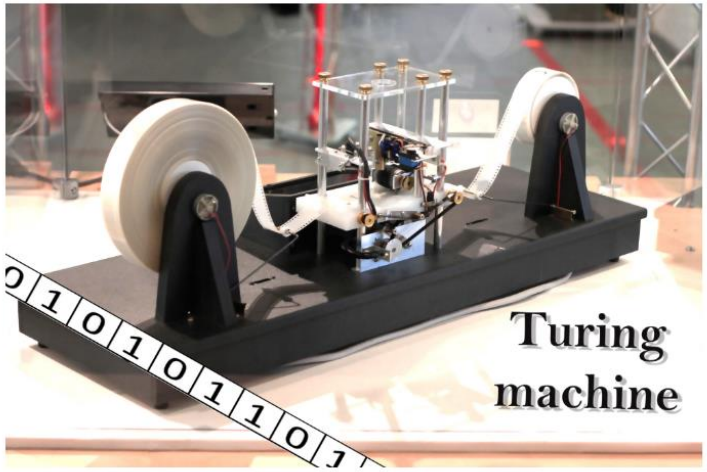
```
  1010
+ 0111
------
 10001
```

RAM
Random Access Memory

Vs

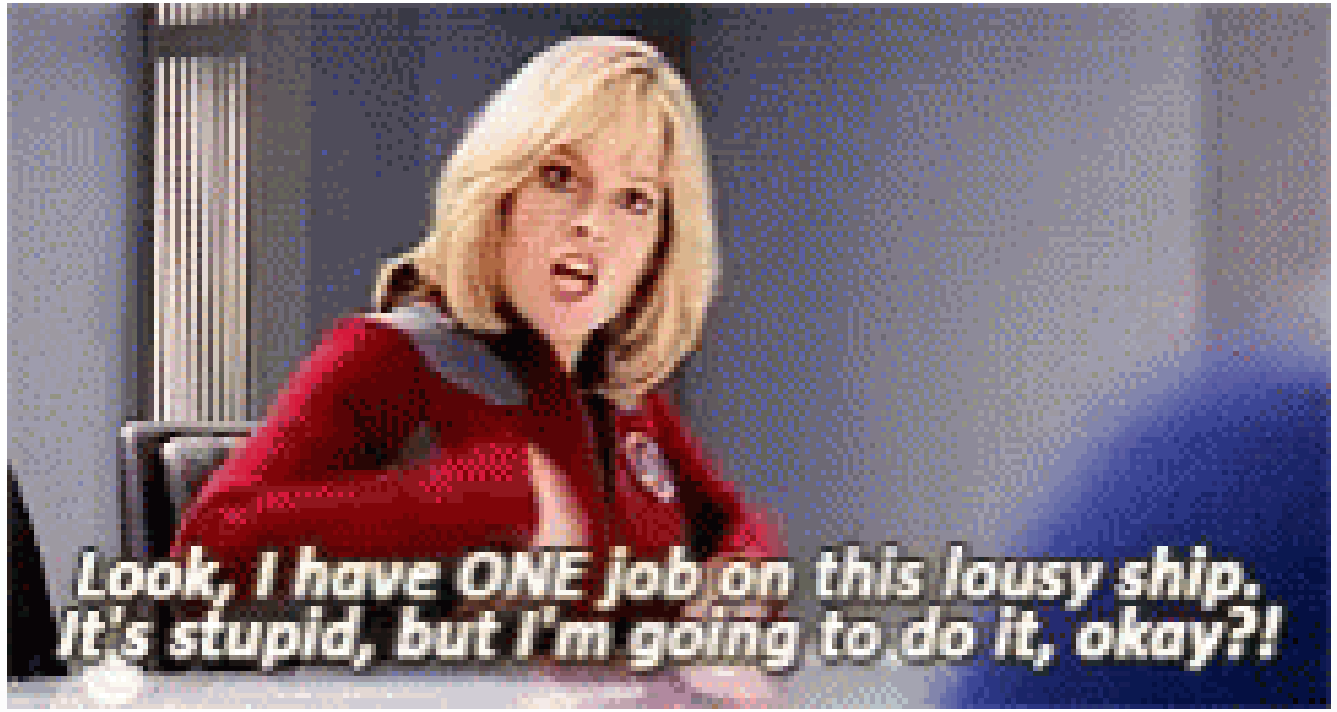


ROM
Read only Memory



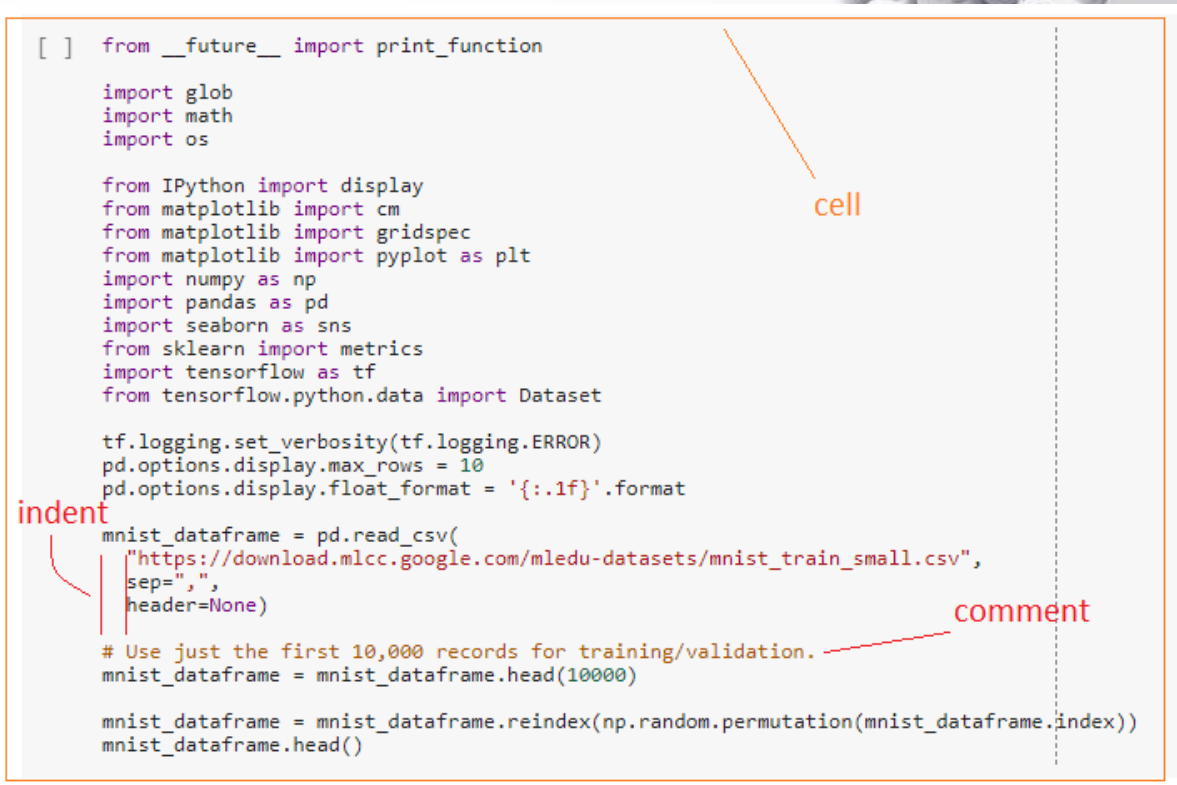
Ultimate Computer Programming

Galaxy Quest



Python Programs

- Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation.
- Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional.



Type in Python (1)

- Python has a dynamic typed.

Type	Mutability	Description	Syntax example
`bool`	immutable	Boolean value	`True` `False`
`bytearray`	mutable	Sequence of bytes	`bytearray(b'Some ASCII')` `bytearray(b"Some ASCII")` `bytearray([119, 105, 107, 105])`
`bytes`	immutable	Sequence of bytes	`b'Some ASCII'` `b"Some ASCII"` `bytes([119, 105, 107, 105])`
`complex`	immutable	Complex number with real and imaginary parts	`3+2.7j`
`dict`	mutable	Associative array (or dictionary) of key and value pairs; can contain mixed types (keys and values), keys must be a hashable type	`{'key1': 1.0, 3: False}`
`ellipsis` [a]	immutable	An ellipsis placeholder to be used as an index in NumPy arrays	`...` `Ellipsis`

Type in Python (2)

- Python has a dynamic typed.

Type	Mutability	Description	Syntax example
`float`	immutable	Floating point number, system-defined precision	`3.1415927`
`frozenset`	immutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	`frozenset([4.0, 'string', True])`
`int`	immutable	Integer of unlimited magnitude[87]	`42`
`list`	mutable	List, can contain mixed types	`[4.0, 'string', True]`
`NoneType` [a]	immutable	An object representing the absence of a value.	`None`
`NotImplementedType` [a]	immutable	A placeholder that can be returned from overloaded operators to indicate unsupported operand types.	`NotImplemented`
`set`	mutable	Unordered set, contains no duplicates; can contain mixed types, if hashable	`{4.0, 'string', True}`
`str`	immutable	A character string: sequence of Unicode codepoints	`'Wikipedia'` `"Wikipedia"` `"""Spanning multiple lines"""`
`tuple`	immutable	Can contain mixed types	`(4.0, 'string', True)`

Basics(1)

```
[ ]  three_type_of_energy = ["protein", "carbohydrates", "fat"]
```

Multiple procedural statements

```
[ ]  protein, carbohydrate, fat = three_type_of_energy
     print(f"{carbohydrate} sure taste good")
     print(f"{fat} isn't bad for you anymore?")
```

```
carbohydrates sure taste good
fat isn't bad for you anymore?
```

Adding Numbers

```
[ ]  protein = 4
     fat = 9
     carbohydrate = 4
     carbohydrate + protein
```

```
8
```

Adding Phrases

```
[ ]  "a carbohydrate " + "has " + str(carbohydrate) + " calories"
```

```
'a carbohydrate has 4 calories'
```

Basics(2)

dict

```
[ ]  omelette = {"egg": 3, "ham": "yes"}
     type(omelette)
```

```
dict
```

list

```
[ ]  ingredients = ["egg", "ham", "bacon"]
     type(ingredients)
```

```
list
```

set

```
[ ]  egg_set = set(["egg", "egg"])
     type(egg_set)
```

```
set
```

```
[ ]  egg_set
```

```
{'egg'}
```

tuple

```
[ ]  breakfast = ("egg","soup")
     breakfast[0] = "turkey"
```

Basics(3)

```
[ ]  import math
     math.pow(2,3)
```

```
8.0
```

Can also use built in exponent operator to accomplish same thing

```
[ ]  2**3
```

```
8
```

multiply

```
[ ]  2*3
```

```
6
```

this is regular multiplication

```
[ ]  2*3
```

```
6
```

Converting Between different numerical types

There are many numerical forms to be aware of in Python. A couple of the most common are:

- Integers
- Floats

```
[ ]  number = 100
     num_type = type(number).__name__
     print(f"{number} is type [{num_type}]")
```

```
100 is type [int]
```

Basics(4)

break

```
[ ]  carbohydrate = 4
     calories = 0
     while True:
       calories += carbohydrate
       print(f"Eating more carbohydrates {calories}")
       if calories > 8:
         print("This is all I can eat")
         break
```

```
Eating more carbohydrates 4
Eating more carbohydrates 8
Eating more carbohydrates 12
This is all I can eat
```

```
[ ]
```

continue

```
[ ]  three_type_of_energy = ["protein", "sugar", "fat"]
     for energy in three_type_of_energy:
       if energy == "sugar":
         print(f"skipping {energy} for my health")
         continue
       print(f"eating {energy}")
```

```
eating protein
skipping sugar for my health
eating fat
```

Index & Slicing

Index and Slicing

```
[ ]  nums = list(range(5))      # range is a built-in function that creates a list of integers
     print(nums)                # Prints "[0, 1, 2, 3, 4]"
     print(nums[2:4])           # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
     print(nums[2:])            # Get a slice from index 2 to the end; prints "[2, 3, 4]"
     print(nums[:2])            # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
     print(nums[:])             # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
     print(nums[:-1])           # Slice indices can be negative; prints "[0, 1, 2, 3]"
     nums[2:4] = [8, 9]         # Assign a new sublist to a slice
     print(nums)
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

```
[ ]  numbers = [2,3,4,5,6,7,8,9,10]
     print(numbers[0], numbers[-1], numbers[-3])
     print(numbers[:])
     print(numbers[0:3], numbers[0:], numbers[:3], numbers[:-3])
```

Control Structure

```
[ ]  # for loop
     for n in range(100):
       print(n)
```

```
[ ]  S = "this is a string"
     for s in S:
       print(s)
```

```
[ ]  # It is not preferred to write a while true
     x = 0
     while True:
       print(x)
       x += 1
       if (x==5):
         break
```

```
[ ]  for n in range(2, 10):
       for x in range(2, n):
         if n % x == 0:
           print(n, 'equals', x, '*', n//x)
           break
         else:
           # loop fell through without finding a factor
           print(n, 'is a prime number')
```

```
[ ]  for num in range(2, 10):
       if num % 2 == 0:
         print("Found an even number", num)
         continue
       print("Found a number", num)
```

Functions

```
[ ]  def function1():
         print('Do something here')

     function1()
```

```
[ ]  x = 1

     def function2(y):
         x = 2
         print('x = ', x + y)

     function2(3)
```

```
[ ]  def CalAreaOfACircle(r):
         # this is a comment
         return np.pi*r*r

     CalAreaOfACircle(4)
```

```
[ ]  a = function2(3)
     b = CalAreaOfACircle(1)
     print(a," and ",b)
```

```
[ ]  x = lambda a : a + 10
     print(x(5))
```

Class

```
[ ]  class A:
       var1 = 12
       var2 = 'bars'

     a1 = A()
     a2 = A()
     print(a1.var1, a1.var2)
     print(a2.var1, a2.var2)
     a1.var1 = 'foo'
     a1.var2 = 12
     print(a1.var1, a1.var2)
     print(a2.var1, a2.var2)
```

```
[ ]  class B:
       def __init__(self):
         self.var1 = 12
         self.var2 = 'bars'

       def boo(self):
         print(self.var1+1,'foo'+self.var2)

     b1 = B()
     print(b1.var1, b1.var2)
     b1.boo()
```

```
[ ]  class C(B):
       def __init__(self):
         super().__init__()
         self.var3 = 144
         self.var4 = 'bottles'

       def boom(self):
         print(self.var3,self.var4)
```

Import Modules

```
[ ]  import math
     print( abs(-10), max([1,2,3,4,3,2,1]), min([1,2,3,4,3,2,1]) )
     print( math.e, math.pi )
     print( math.ceil(12/5), math.floor(12/5), math.exp(1), math.log(math.exp(10)), math.log10(10) )
     print( math.pow(2,3), math.sqrt(16) )
```

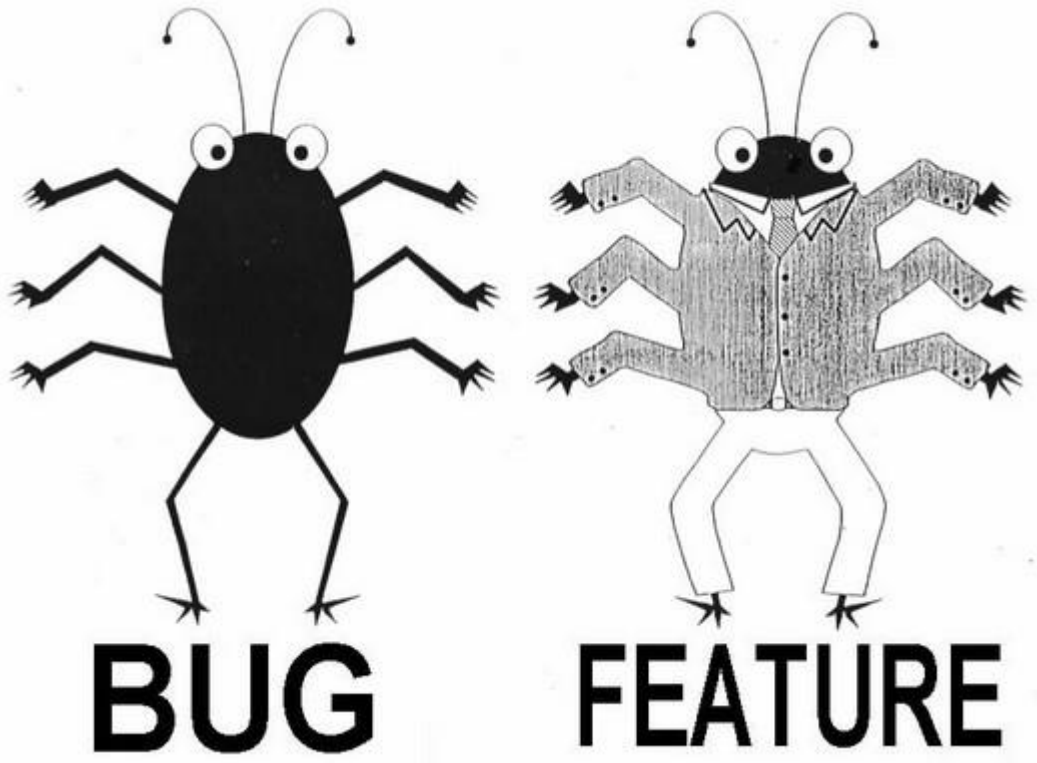
```
10 4 1
2.718281828459045 3.141592653589793
3 2 2.718281828459045 10.0 1.0
8.0 4.0
```

```
[ ]  print(math.sin(0))
     print(math.cos(0))
     print(math.sin(math.pi))
     print(math.cos(math.pi/2))
     print(math.sin(math.pi/2))
     print(math.cos(math.pi))
```

```
0.0
1.0
1.2246467991473532e-16
6.123233995736766e-17
1.0
-1.0
```

```
[ ]  import random
     print( random.random() )
     print( random.choice([1,2,3,4,5]) )
     print( random.randrange (-5,6) )
     a = [1,2,3,4,5]
     print( random.shuffle(a),a )
```

Programming Exercises



Practicals with Jupyter Notebooks

- Using Python as a programmable calculator.
- Using Python as a programming language.

Q & A

https://data-flair.training/blogs/