## Chapters 0, 1, 2. Variables and Data Types

```python
import numpy as np

me = 9.11e-31      # mass of electron
c  = 299792458     # speed of light

u  = 0.1 * c       # particle velocity

gamma = 1 / np.sqrt(1-(u/c)**2)   # gamma factor

KE = (gamma-1) * me * c**2        # relativistic kinetic energy
```

# Python for Physicists

# Python For Physicists

## Chapters 0-2: Introduction, Variables and Data Types

# Intro to Python

- Python is a widely used programming language that's popular in science, data analysis, and everyday problem-solving because it's designed to be readable and **easy to learn**.
- In Python, you can use it like a calculator to do math, work with text, or organize data into lists and tables.
- In Python, **every piece of data comes with built-in tools for working with it** (this is true for numbers, characters, etc), so you can quickly get things done without starting from scratch. For the experts:  everything in Python is an **object** with pre-defined **attributes** and **methods**.
- As you learn Python, you'll see how a few simple building blocks can be combined to model real-world problems, run experiments, and solve practical tasks.

# Your First Program

- The following program displays a message to the screen. The output is shown to the right

```
print(“hello world”)
```

```
hello world
```

- Strings (e.g. “hello world”) come with a set of **built-in abilities** or tools (which we call **attributes** and **methods**).

- For example, strings “know how” to make themselves upper-case:

```
print(“hello world”.upper())
```

```
HELLO WORLD
```

# Variables

- Variables are labels used to reference values stored in computer memory
- Variable names can only use letters, numbers and the underscore “_”
- Variable names can’t start with a number
- Python is case-sensitive ( “X” is not the same as “x”)

**Valid variable names:**

```
mass = 10
m = 20
first_name = “Sal”
g = 9.8
```

**invalid names:**

```
2mass = 10
#mass = 20
first name = “Sal”
```

# Use variables to perform calculations

- We define two variables to store the length and width of a rectangle
- We calculate and display the area to the screen
- Notice we also use comments (defined by #) to remind ourselves the measurement is in meters. Python variables do not have units. You need to need track them yourself.

```
length = 10                    # in meters
width = 20                     # in meters
area = length * width          # in meters^2
print(“area =”, area, “m^2”)
```

```
area = 200 m^2
```

- Notice also, we can display a combination of variables and text in the print statement by separating them with commas

# Data Types

- Values (like numbers and text) have different **data types**.

**Examples:**

| | |
|---|---|
| Integer | 4 |
| Float | 4.2 |
| Complex | 4.2+2j |
| String | “hello” |
| Boolean | True |

# Data Types

- Different data types can behave differently under the same operation

**Example:** The * operator generates repeated copies of a string

```
s = “4”          # string
print(10*s)
```

```
4444444444
```

but performs mathematical multiplication on integers and floats

```
s = 4            # integer
print(10*s)
```

```
40
```

# Data type conversions

**To:**

| **From:** | **Integer** | **Float** | **String** |
|---|---|---|---|
| Integer | | float(4) | str(4)<br>f'{x:i}' |
| Float | int(4.0)<br>round(4) | | str(4.0)<br>f'{x:.2f}' |
| String | int("4") | float("4.0") | |

**Examples:**

```
pi = 3.14153
x  = 4
s  = "50"
```

```
y = int(pi)       # y = 3
y = float(s)      # y = 50.
z = str(x)        # z = "4"
y = f'{pi:.2f}'   # y = "3.14"
```

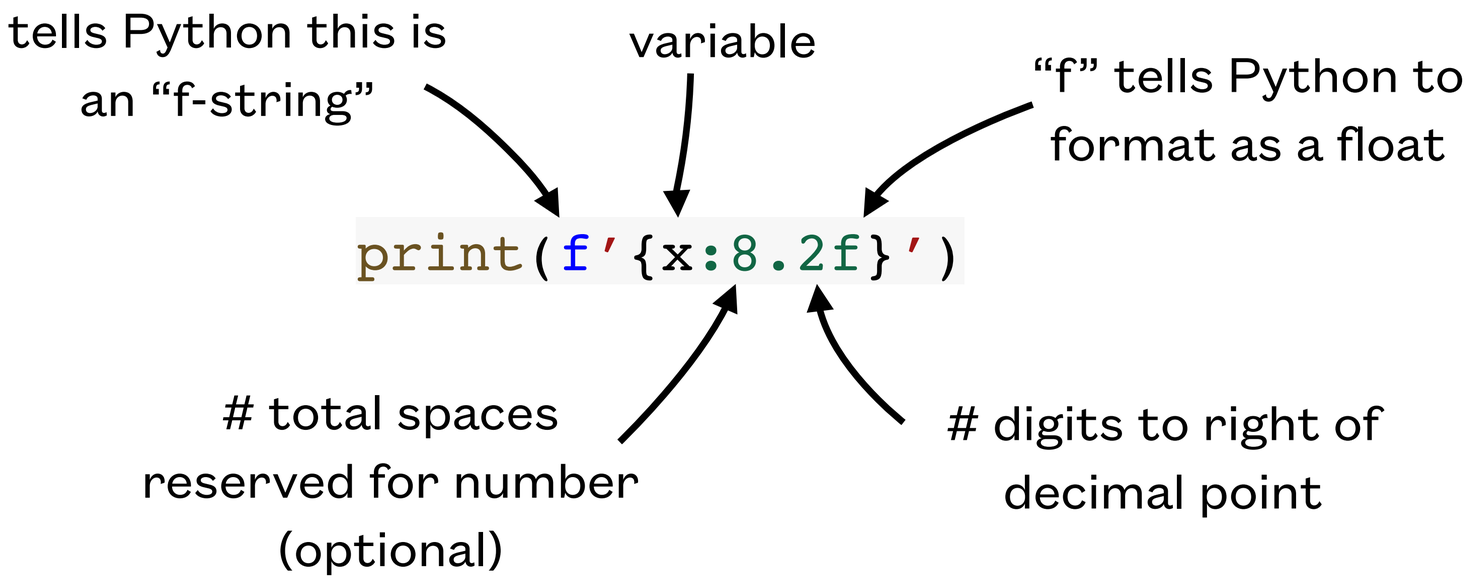
float → int
string → float
int → string
float → string

# Formatted printing with f-strings

- Formatted printing lets you line things up in neat columns and control the number of significant digits

  - Formatting floats:

```
x = 316.227766  # define a float
```




Output: 316.23

# Formatted printing with f-strings

- f-string options:

```
x = 316.227766           # define a float
n = 2478                 # define a float

print(f'{x:.2f}')        # 316.23          "f" = decimal representation
print(f'{x:.4f}')        # 316.2278
print(f'{x:8.2f}')       #   316.23
print(f'{x:10.2f}')      #     316.23

print(f'{x:.2e}')        # 3.16e+02        "e" = exponential (i.e. scientific)
                         #                       notation

print(f'{n:6g}')         #   2478          "g" = chooses decimal or scientific
                         #                       notation to make easy to read

print(f'{n:6d}')         #   2478          "d" = integer representation
```

# Errors

- Believe it or not, error messages are your friends!  They help you find bugs in your code.

```
print("hello)
```

```
  File "/tmp/ipython-input-2149988476.py", line 1
    print("hello)
          ^
SyntaxError: unterminated string literal (detected at line 1)
```

```
float("cat")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-658078711.py in <cell line: 0>()
----> 1 float("cat")

ValueError: could not convert string to float: 'cat'
```

# Errors

```
print("hello)
```

```
  File "/tmp/ipython-input-2149988476.py", line 1
    print("hello)
          ^
SyntaxError: unterminated string literal (detected at line 1)
```

When you get an error, do one or more of the following:

- don’t panic
- don’t throw objects
- take a deep breath
- relax
- meditate
- give thanks to the error message for helping you make your code the best it can be
- use the line number to identify where the error is.
- sometimes copying the error into ChatGPT or another AI can help with troubleshooting

# Python For Physicists

## Chapters 3:  Math Functions

# Native Python has a few built-in Math functions

```
x = 5           # assign a value to x
y = 2           # assign a value to y

z1 = x + y      # adds x and y
z2 = x - y      # subtracts y from x
z3 = x * y      # multiplies x times y
z4 = x / y      # divides x by y

x += 5          # adds 5 to the current value of x
x -= 5          # subtracts 5 to the current value of x
x *= 5          # multiplies the current value of x by 5
x /= 5          # divides the current value of x by 5

abs(x)          # takes the absolute value of x
round(x)        # rounds x to nearest integer
int(x)          # truncates decimal leaving only whole number

x % y           # remainder after division of x / y
x // y          # integer division of x / y (truncates decimal)
```

# Some function have default values for some arguments:

Rounds to the nearest integer

```
x = round(3.144159)
```

```
x → 3
```

Rounds to the nearest N decimal places

```
N = 2
x = round(3.144159,N)
```

optional argument

```
x → 3.14
```

# Define your frequently used physical constants

```
c     = 299792458          # definition of the speed of light in m/s
h     = 6.626e-34          # Planck's constant (J s)
hbar  = 1.0546e-34         # "h bar" = h / (2*pi)  (J s)
k     = 1.3806e-23         # Boltzmann's constant (J/K)
G     = 6.6743e-11         # Gravitational constant (m^3/kg/s^2)
e     = 1.602177e-19       # fundamental charge (C)
me    = 9.10938e-31        # mass of electron (kg)
u     = 1.66054e-27        # atomic mass unit (kg)

epsilon0 = 8.854188e-12  # vacuum permittivity (F/m)
```

# NumPy Library

Libraries are collections of one or more modules (collections of functions and other objects) that provide additional functionality to the basic Python package, much like a new instrument adds functionality to a research lab.

One of the most widely-used libraries for scientific computation is **NumPy** (pronounced "num-pie" and not something that rhymes with "grumpy").

- NumPy provides a wide range of numerical functions that are not included in native Python.
- For example, Python doesn't natively include the sine and cosine functions.
- To use this library, we add an import statement to the beginning of our program, and give it a nickname:

```
import numpy as np    # import the numpy library and give it nickname  “np”

r = 10                # define the radius of a circle
A = np.pi * r**2      # use the numpy library value for pi
print("area of circle = ",A)
```

# Commonly used NumPy math functions

```
x = 0.5    # define a value for x
y = 3      # define a value for y

# constants
np.pi          # pi
np.e           # e
np.inf         # infinity
np.nan         # not a number

# logarithmic and exponential functions
np.sqrt(x)     # square root(x)
np.exp(x)      # e^x
np.log(x)      # ln(x)
np.log10(x)    # log base 10(x)
np.log2(x)     # log base 2(x)

# trigonometric functions
np.sin(x)      # sin(x)
np.cos(x)      # cos(x)
np.tan(x)      # tan(x)
```

```
# degree-radian conversions
np.deg2rad(x)      # converts degrees to radians
np.rad2deg(x)      # converts radians to degrees

# inverse trigonometric functions
np.arcsin(x)  # asin(x)
np.arccos(x)  # acos(x)
np.arctan(x)  # atan(x)

# hyperbolic functions
np.sinh(x)     # hyperbolic sin
np.cosh(x)     # hyperbolic cos
np.tanh(x)     # hyperbolic tan
```