# ENGR 21: Computer Engineering Fundamentals

Instructor: Emad Masroor

Lecture 4
Thursday, September 11, 2025

# Python Clinic for E21

## for beginner programmers

by Nick Fettig and Owen Hoffman, Class of 2026

Sunday, Sep 14
7 - 9 PM
Location TBD

# Logical operators and comparison operators in Python

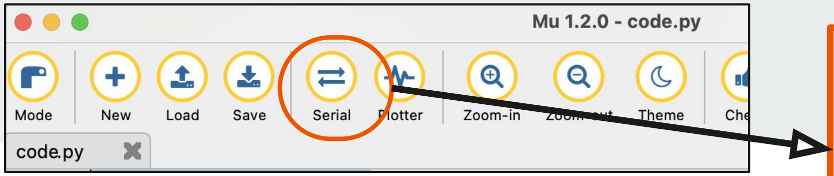
Logical ops.

| | | |
|---|---|---|
| ‘And’ | **and** | **&** |
| ‘Or’ | **or** | **\|** |
| ‘Not’ | **not** | |

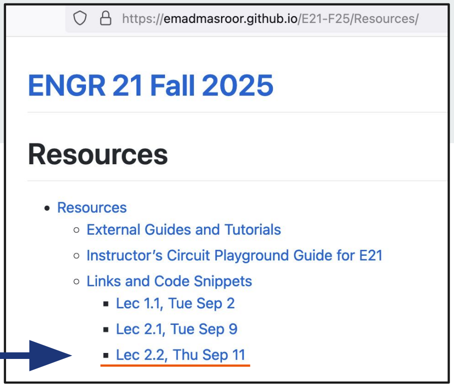
Comparison ops.

| | |
|---|---|
| Equals | **==** |
| Does not equal | **!=** |
| Greater than | **>** |
| Less than | **<** |
| Greater than or equal to | **>=** |
| Less than or equal to | **<=** |

```
>>> 2 < 3 and 4 < 3
False

>>> 2 < 3 or 10 < 9
True

>>> not True
False
>>> not False
True


>>> 2 < 3 or 5 < 6 and 3 == 4

>>> 2 < 3 or (5 < 6 and 3 == 4)

>>> (2 < 3 or 5 < 6) and 3 == 4

>>> 2 < 3 or ('a' < 'b' and 'a' < 'A')

>>> (2 < 3 or 'a' < 'b') and 'a' < 'A'
```



copy code
from here
into code.py !



ENGR 21 Fall 2025

Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11

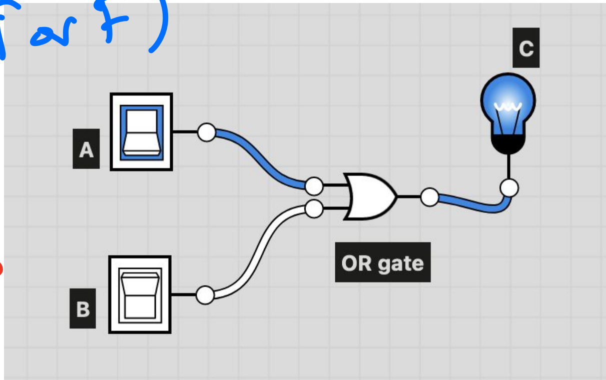
# Truth Tables & Nested If statements

- A and B are connected via an OR gate to C

boolean (T or F)

C = A or B

also booleans



| A | B | C |
|---|---|---|
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

implementation of logic gate C

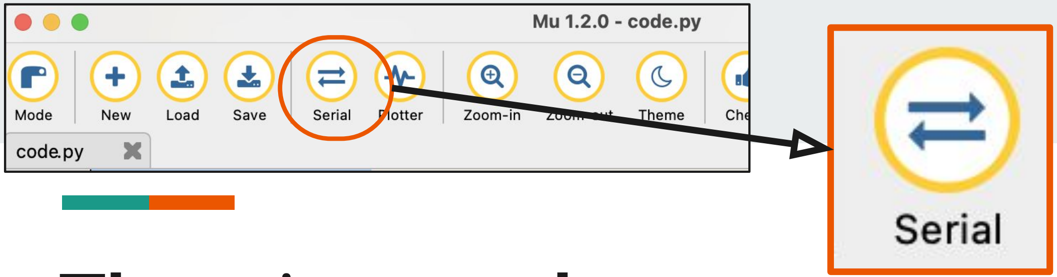
'C = A or B'

```
# Set the values of A and B
A = True
B = False

# Implement "Logic gate OR" by covering
all four possibilities.

if A == True:
    if B == True:
        C = True # line 1 of table
    else:
        C = True # line 2 of table
else:
    if B == True:
        C = True # line 3 of table
    else:
        C = False # line 4 of table

print("--After applying logic, C is",C)
```

→ set to false

→ false

then C = false



# The `time` package

This package is built into Circuit Python and allows you to **time actions** on the Circuit Playground Bluefruit.

To use: add this line to the top of **`code.py`**

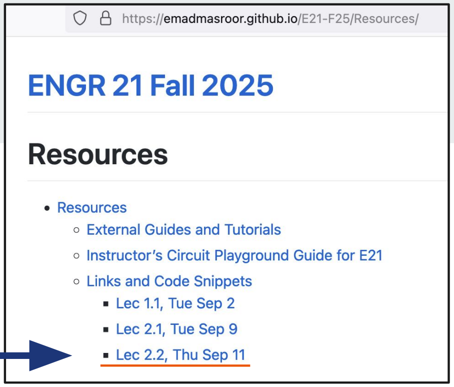
```
import time
```

For now, we will use just one feature of this:

```
time.sleep(x)
```

Where x is the number of seconds you want to pause.

copy code from here into code.py !



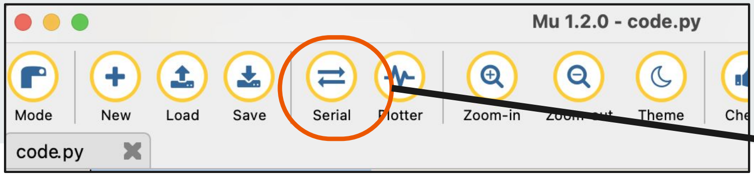
https://emadmasroor.github.io/E21-F25/Resources/

## ENGR 21 Fall 2025

### Resources

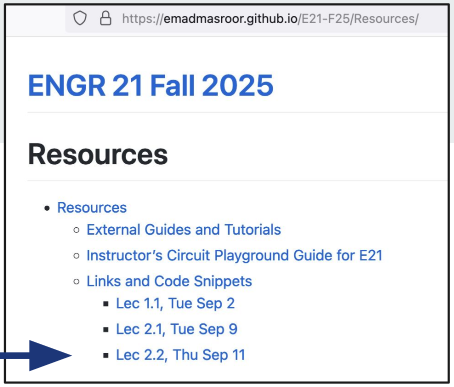
- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11

```
from adafruit_circuitplayground import cp

import time

delay = 3.0

print("Switching on pixel 0")
cp.pixels[0] = (0,10,0)
print(f"Waiting for {delay} seconds")
time.sleep(delay)

print("Switching on pixel 1")
cp.pixels[1] = (10,0,0)
print(f"Waiting for {delay} seconds")
time.sleep(delay)

print("Switching on pixel 2")
cp.pixels[2] = (0,0,10)
print(f"Waiting for {delay} seconds")
time.sleep(delay)
```



# Printing with variables

copy code from here into code.py !


https://emadmasroor.github.io/E21-F25/Resources/

## ENGR 21 Fall 2025

### Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11

Print statements send text to the console.

```
print(“There are 12 units in a dozen”)
```

how do we make this a variable?

```
print("The number is 24")

p = 24.1
n = 24

print("The numbers are n and p")          → X
print("The numbers are {n} and {p}")      → X
print(f"The numbers are {n} and {p}")
print(f"The numbers are {n:} {p:.3f}")
c = "The numbers are {} and {}".format(p,n)
print(c)
```

print(f"your string {var} text")

optional : $\{var:\underbrace{.4f}_{\text{4 digits after decimal point}}\}$

if you want to specify precision.

4 digits after decimal point

# `for` and `while` loops in Python

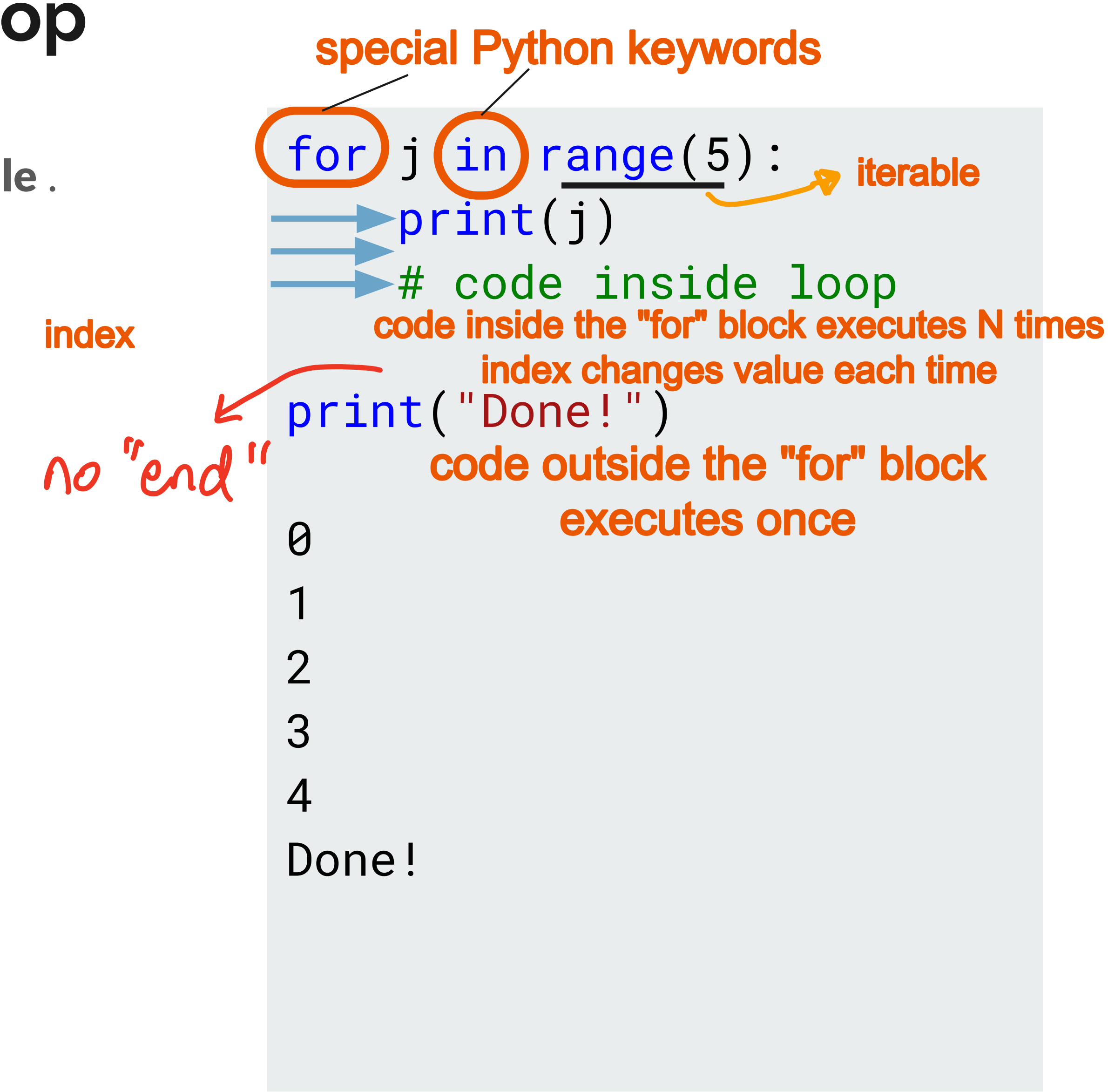
(Later we will do functions…)

# Anatomy of a `for` loop

A `for` loop "iterates over" an **iterable** .

Iterables in Python:

- List
- Tuple
- Range
- Strings
- Also:
  - Dictionaries
  - Sets



special Python keywords

```
for j in range(5):
    print(j)
    # code inside loop
print("Done!")
```

iterable

index

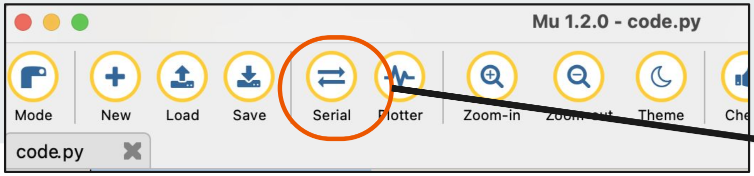
code inside the "for" block executes N times

index changes value each time

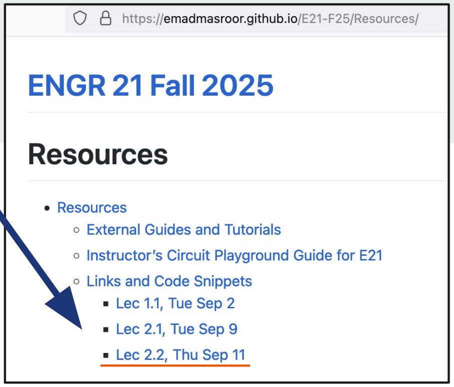
no "end"

code outside the "for" block executes once

```
0
1
2
3
4
Done!
```



copy code from here into code.py !



https://emadmasroor.github.io/E21-F25/Resources/

ENGR 21 Fall 2025

Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11

# Using `for` loops with various iterables

→ Range doesn't have to be from zero, and doesn't have to increment by 1.

→ Iterate through characters of a string.

alternatively:

```
c = "hello"
for j in range(5):
    print(c[j])
```

jth index of c

```
# Iterable (1): Range
print("Printing fron a range:")
for j in range(2,10,2):
    print(j)

# Iterable (2): list
print("Printing fron a list:")
a = [1,"a",6,"hello",5,True]
for j in a:
    print(j)

# Iterable (3): tuple
print("Printing fron a tuple:")
b = (1,2,"x",3,1)
for k in b:
    print(k)

# Iterable (4): string
print("Printing characters from a string:")
c = "hello"
for x in c:
    print(x)
```

# Anatomy of a `while` loop

A `while` loop runs “while” a conditional is true.

- Any Python conditional can be used
- Conditional is checked **every** loop iteration, at the start.

```
while 3>2
while True
```

} equivalent.

```
for j in range(10) :
    if (...check if j is odd) :
        do something.
```

conditional whose truth is checked each iteration.

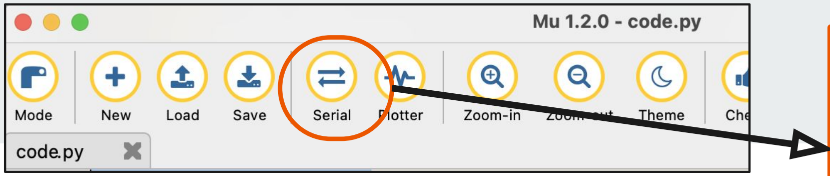
special Python keyword

```
while 3 > 2:
    print("hello!")
```

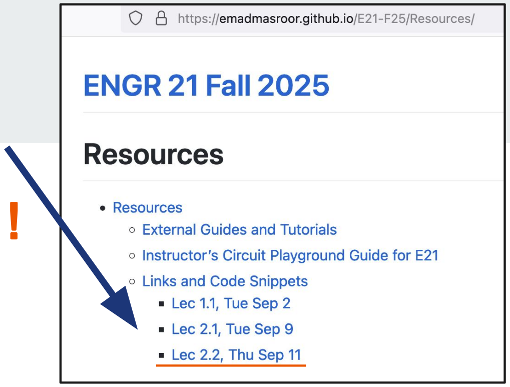
code inside the "while" block runs while conditional is True

```
hello!
hello!
hello!
hello!
hello!
hello!
..
```

```
k=0
while k < 10 :
    if (check k odd):
        do something
    k=k+1
```



copy code from here into code.py !



https://emadmasroor.github.io/E21-F25/Resources/

ENGR 21 Fall 2025

Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11

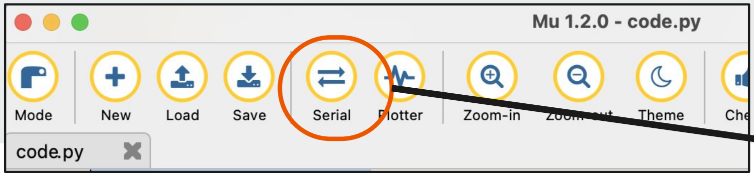
# The `break` keyword

- Exits the loop
  - `for`: Regardless of whether all elements of the iterable have been traversed
  - `while`: Regardless of whether the conditional is still true.
- Works with `for` and `while` loops.

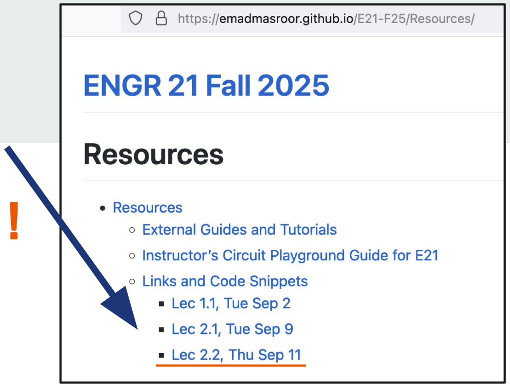
```
# break inside for loop
for j in range(10):
    print(j)
    if j == 3:
        print("exiting loop")
        break

0
1
2
3
exiting loop
```

k=k+1 } equiv.
k+=1

```
# break inside while loop
counter_variable = 0
while 3 > 2:
    counter_variable += 1
    if counter_variable > 3:
        break
0
1
2
3
```



copy code from here into code.py !



https://emadmasroor.github.io/E21-F25/Resources/

ENGR 21 Fall 2025

Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11

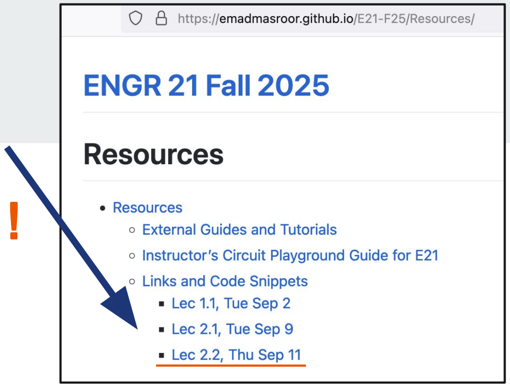
# The `continue` keyword

- Exits the current iteration
- The next iteration continues as usual.
- Works with `for` and `while` loops.

```
for j in range(5):
    print(j)
    if j == 2:
        continue
    print("iteration complete",j)
```

# Task: Light up pixels using `for` and `while` loops

copy code from here into code.py !



https://emadmasroor.github.io/E21-F25/Resources/

## ENGR 21 Fall 2025

### Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11

Write your own code inside `code.py` that:

- Lights up each NeoPixel for 1 second, in order
- Each pixel should be brighter than the last by 20 units

Using one of 3 different techniques:

- ○ a **`while`** loop
- ○ a **`for`** loop that uses **`range(10)`**
- ○ a **`for`** loop that uses **`range(10,200,20)`**

```
from adafruit_circuitplayground import cp
import time

cp~~x~~.pixels[0] = (10,10,10)
time.sleep(1)
cp~~x~~.pixels.fill((0,0,0))

cpx.pixels[1] = (30,30,30)
time.sleep(1)
cp~~x~~.pixels.fill((0,0,0))

cp~~x~~.pixels[2] = (50,50,50)
time.sleep(1)
cp~~x~~.pixels.fill((0,0,0))

cp~~x~~.pixels[3] = (70,70,70)
time.sleep(1)
cp~~x~~.pixels.fill((0,0,0))
```

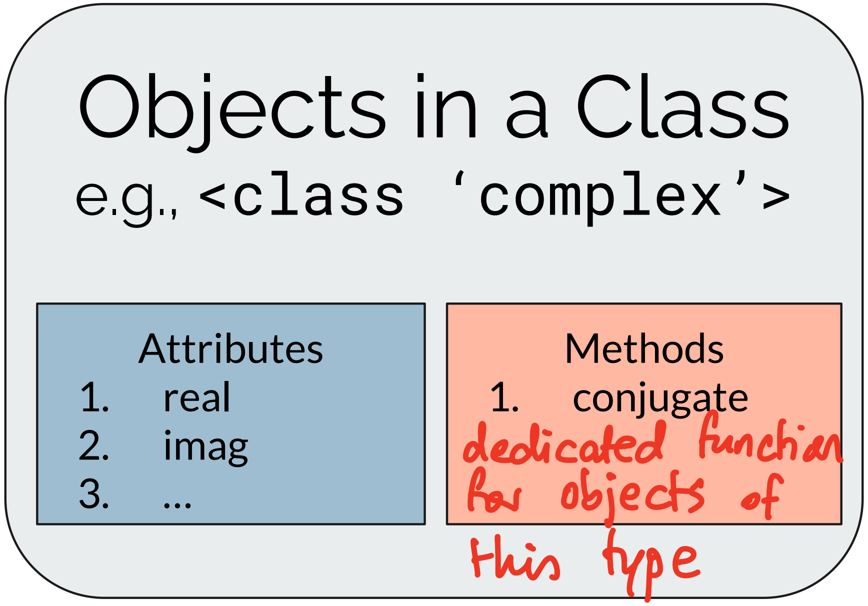
Loops are an efficient way to write code like this so you don't have to re-write the same thing

# Objects in Python

# Objects, classes and methods in Python

Python is an object-oriented language

```
>>> z1 = 4 + 5j
>>> type(z1)
<class 'complex'>

>>> z1.real
4.0

>>> z1.imag
5.0

>>> complex.conjugate(z1)
(4-5j)

>>> z1.conjugate()
(4-5j)
```



Objects in a Class

e.g., `<class ‘complex’>`

Attributes
1. real
2. imag
3. …

Methods
1. conjugate

dedicated function for objects of this type

Try this code in the REPL!

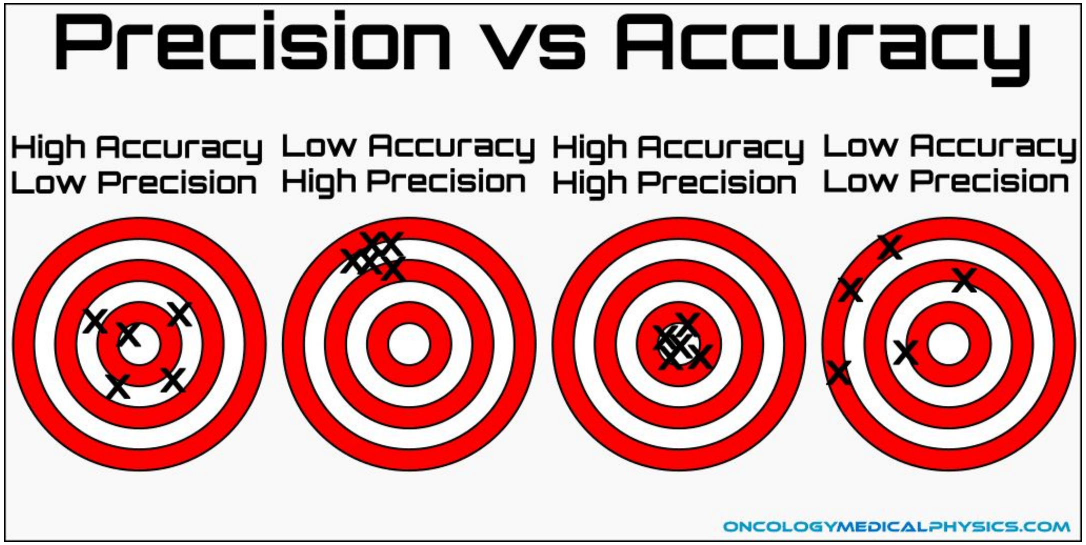
- ‘`z1`’ is an instance of class `complex`
- ‘`z1`’ is an object of type `complex`
- ‘`z1`’ has attributes `real` and `imag`
- ‘`z1`’ has a method `conjugate`

# Errors, precision and accuracy

# Precision vs Accuracy: What's the difference?

"Consistency"

how close to true value


Precision vs Accuracy
High Accuracy Low Precision
Low Accuracy High Precision
High Accuracy High Precision
Low Accuracy Low Precision
ONCOLOGYMEDICALPHYSICS.COM

# Ways of quantifying error

When you know the true value:

- Absolute error $\left| \text{True value} - \text{Measured value} \right|$

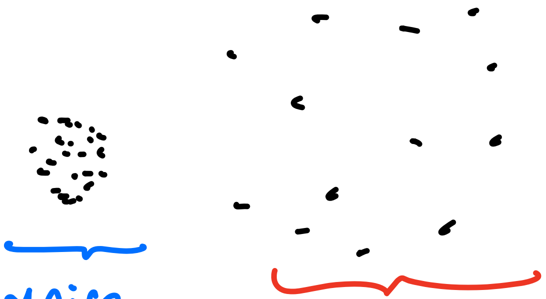
- $\underbrace{\text{Relative error}}_{\text{"relative to" the true value}}$ $\dfrac{\left| \text{True value} - \text{Measured value} \right|}{\left| \text{True value} \right|}$

# Ways of quantifying precision

Regardless of the “true” value

→ Std is a measure of precision.

→ “Range” i.e
Max − Min
etc.



precise
low standard deviation

imprecise
high standard deviation