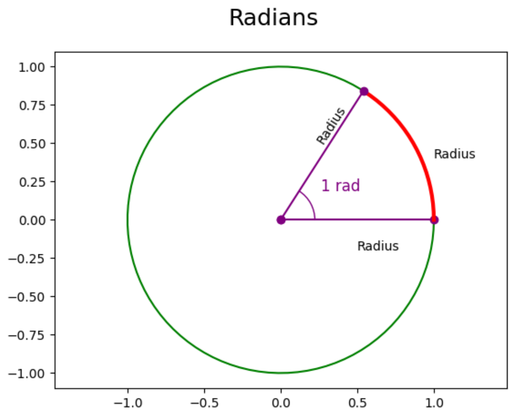
Measuring Angles: Degrees vs Radians

You're used to measuring angles in terms of **degrees**. There are other units of angular measurement:

- **degrees** : A degree is 1/360 of a circle. A quarter of a circle is 90°
- **turns** : There is 1 turn in a complete circle. A quarter of a circle is 0.25 **turns**
- **radians** : A radian is the angle of an arc the length of the circle's radius. There are 2π radians in a circle. A quarter of a circle is π/2 **radians**.



In programming, radians are more commonly used than degrees

In many programming libraries, including Python's `math` and `numpy` modules, radians, not degrees, are used to measure angles.

Q: Why are radians preferred? A: ***radians*** *and* ***turns*** *are more "natural" ways of expressing "how far around a circle are we?" and "how far along in a cycle are we?".*

*Dividing a circle into 360 parts is kind of arbitrary: why 360 and not, say, 12 or 36 or 720 or 1440? But "a radius worth of arc around a circle" (**radian**) or "percentage of way around a circle" (**turn**) feel a little more natural.*

You can use either radians or degrees, just as you can use either meters or feet, Celsius or Fahrenheit. What's important is that:

- Be consistent. Use radians *or* degrees throughout your programs!
- When you call a function that deals with angles, always double-check that you're passing the angle as measured in the proper unit!

There are 2π radians in a circle and there are 360 degrees in a circle:

$$2\pi \text{ rad} = 360^\circ$$

So:

$$1 \text{ rad} = \frac{360^\circ}{2\pi} = 57.3...^\circ$$

$$1^\circ = \frac{2\pi}{360} = 0.01745...\text{rad}$$

Exercise: Python conversion

Write two functions in Python:

- `to_degrees(rads) : ...` : Returns the number of degrees equivalent to the passed-in `rads` radians value
- `to_radians(degs) : ...` : Returns the number of radians equivalent to the passed-in `degs` degrees value

Test cases:

Degrees	Radians
0	0
360	6.28 (2π)
180	3.14 (π)
90	1.57 (π/2)
45	0.785 ((π/4)
-45	-0.785 (-(π/4))
57.3	1.00001

- Copy the following code into a file `deg_rad_conversion.py` .
- Replace the `return -1` in the `to_degrees` and `to_radians` functions with the proper calculation
- Run the program with `python deg_rad_conversion.py`
- Fix any mistakes until you get 0 errors in your conversion

In [ ]:
```python
from math import pi

def to_degrees(radians):
    # Replace this with your conversion code!
    return -1

def to_radians(degrees):
    # Replace this with your conversion code!
    return -1

degs_and_rads = [
    (0, 0),
    (360, 6.28),
    (180, 3.14),
    (90, 1.57),
    (45, 0.785),
    (-45,-0.785),
    (57.3,1.00001)
]

# Returns True if a and b are within epsilon of each other
def close_enough(a, b, epsilon):
    return abs(a-b) < epsilon

# Initialize counts of mistakes to 0
count_of_incorrect_rad_calcs = 0
count_of_incorrect_deg_calcs = 0

# Loop over every test case
for degs, rads in degs_and_rads:
    # Calculate radians for degrees
    calculated_rads = to_radians(degs)
    # Getting within 0.01 radian is fine
    to_rads_close_enough = close_enough(calculated_rads, rads, 0.01)
    # If failure, output values and increment mistake count
    if not to_rads_close_enough:
        print(f"Expected: {degs} degs -> {rads} rad")
        print(f"Got: {degs} degs -> {calculated_rads:.3f} rad")
        count_of_incorrect_rad_calcs += 1

    # Calculate degrees for radians
    calculated_degs = to_degrees(rads)
    # Getting within 1/2 degree is fine
    to_degs_close_enough = close_enough(calculated_degs, degs, 0.5)
    # If failure, output values and increment mistake count
    if not to_degs_close_enough:
        print(f"Expected: {rads} rad -> {degs} degs")
        print(f"Got: {rads} rad -> {calculated_degs:.3f} degs")
        count_of_incorrect_deg_calcs += 1

print(f"Tested {len(degs_and_rads)} cases.")
print(f"Radian calculations failed: {count_of_incorrect_rad_calcs}")
print(f"Degree calculations failed: {count_of_incorrect_deg_calcs}")
```

In [ ]:
```
```