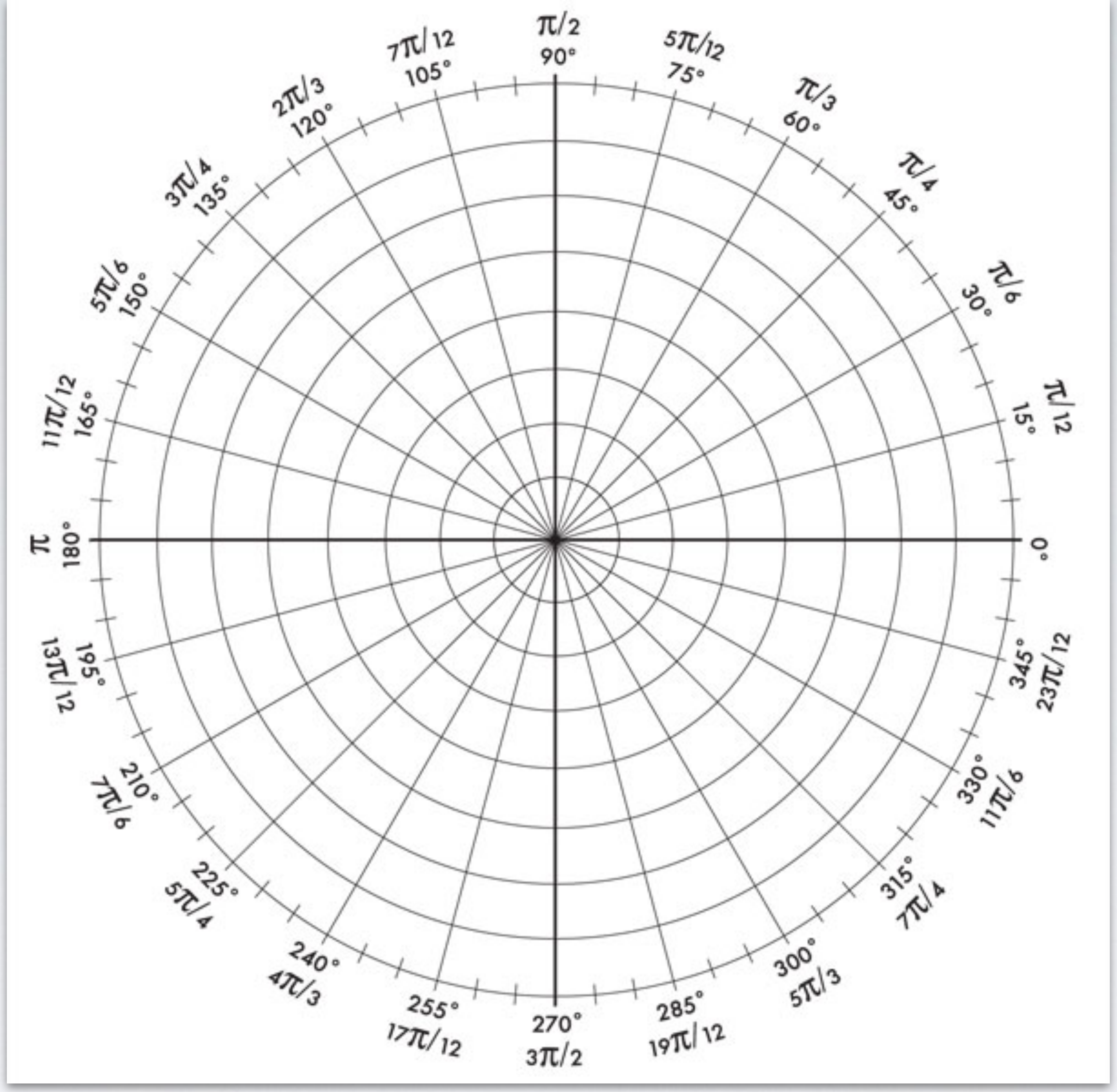
CSCI 135: DIVING INTO THE DELUGE OF DATA

LECTURE 4

functions, conditionals, and modules


π/2
90°
5π/12
75°
π/3
60°
π/4
45°
π/6
30°
π/12
15°
0°
345°
23π/12
330°
11π/6
315°
7π/4
300°
5π/3
285°
19π/12
270°
3π/2
255°
17π/12
240°
4π/3
225°
5π/4
210°
7π/6
195°
13π/12
π
180°
11π/12
165°
5π/6
150°
3π/4
135°
2π/3
120°
7π/12
105°

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

Use the python keyword **def** to define a function

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

polar is the name of the function

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

x and **y** are the function parameters

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

def polar(x, y): is the function header

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

The string following the function header is the **docstring**. It gets bound to the __doc__ method of the **polar** function object

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

The **function body** is a sequence of python expressions. Notice that indentation is significant. All code indented at the same level is part of the same block

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

variables defined within a block are local to that block (they shadow, but don’t destroy variables of the same name in outer blocks but are accessible to inner blocks). These rules mean that Python is a **lexically-scoped** language.

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

functions can be viewed as **procedures**, which abstract away a common set of actions, or as **mathematical functions**, which compute a value. Use return in a function to return a value

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

functions are **called** with (or **applied** to) **arguments.**
The objects assigned to the arguments are passed to the function and bound to the formal parameters.
Here the object assigned to **a** is bound to **x** and the object assigned to **b** is bound to **y**.

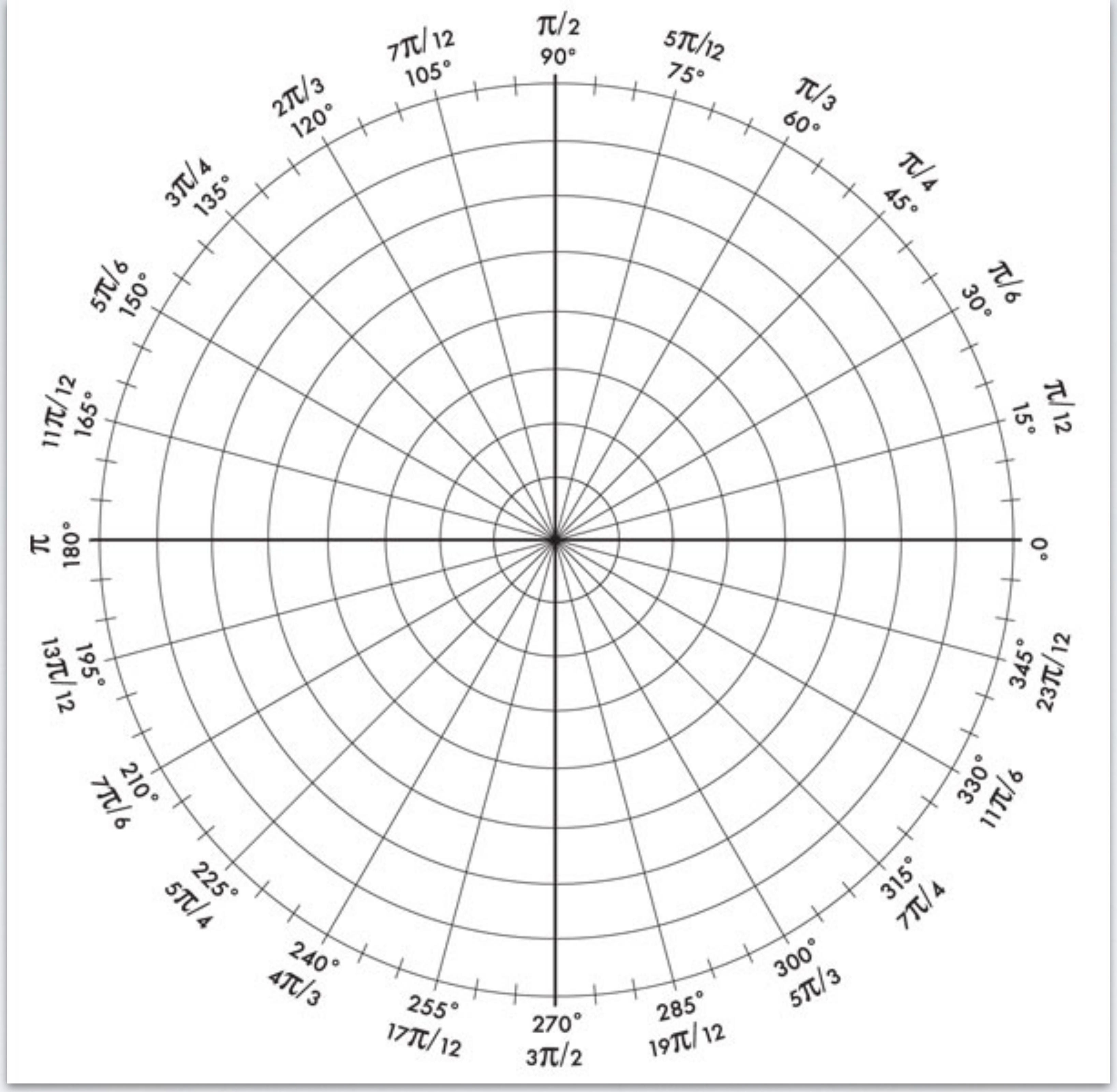
```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

```
>>> a = 1/math.sqrt(2)
>>> b = 1/math.sqrt(2)
>>> polar(a,b)
```

Everything in python is an object. Functions are **function objects** and can be passed as arguments to other functions. When a programming language supports passing functions as first-order objects it is said to support **higher-order functions**.

```
def polar(x, y):
    '''convert (x,y) into polar coordinates
       where the angle is in radians
    '''
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    return (radius, angle)
```

```
>>> a = 1/math.sqrt(2)
>>> b = 1/math.sqrt(2)
>>> polar(a,b)
>>> type(polar)
class <‘function’>
```


π/2
90°
5π/12
75°
π/3
60°
π/4
45°
π/6
30°
π/12
15°
0°
345°
23π/12
330°
11π/6
315°
7π/4
300°
5π/3
285°
19π/12
270°
3π/2
255°
17π/12
240°
4π/3
225°
5π/4
210°
7π/6
195°
13π/12
π
180°
11π/12
165°
5π/6
150°
3π/4
135°
2π/3
120°
7π/12
105°

```
def polar(x, y, deg=False):
    '''convert (x,y) into polar coordinates
       where the angle is in radians (default)
       or degrees (deg=True)
    ‘’'
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    if deg:
        return (radius, angle * 180 / math.pi)
    else:
        return (radius, angle)
```

arguments may have **default values**; arguments without default values cannot appear after arguments with default values

```
def polar(x, y, deg=False):
    '''convert (x,y) into polar coordinates
       where the angle is in radians (default)
       or degrees (deg=True)
    ‘’'
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    if deg:
        return (radius, angle * 180 / math.pi)
    else:
        return (radius, angle)
```

Conditional statements allow you to branch the flow of execution. The **control flow** of **conditional statements** follows the rules of indentation;

```
def polar(x, y, deg=False):
    '''convert (x,y) into polar coordinates
       where the angle is in radians (default)
       or degrees (deg=True)
    ‘’'
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    if deg:
        return (radius, angle * 180 / math.pi)
    else:
        return (radius, angle)
```

the **test** of a conditional statement is a Python expression evaluating to either **True** or **False**; all Python objects have related boolean values; test expressions often involve **equality operation ==**

```
def polar(x, y, deg=False):
    '''convert (x,y) into polar coordinates
       where the angle is in radians (default)
       or degrees (deg=True)
    ‘’'
    radius = math.sqrt(x*x + y*y)
    angle = math.atan2(y, x)
    if deg:
        return (radius, angle * 180 / math.pi)
    else:
        return (radius, angle)
```

- **even(x)** returns **True** if and only if **x** is even
- **odd(x)** returns **True** if and only if **x** is odd
- **min(x,y)** returns the smaller of **x** and **y**
- **max(x,y)** returns the larger of **x** and **y**
- **perfect_square(x)** returns **True** if and only if **x** is a perfect square (i.e. its square root is an integer)
- **fact(x)** returns **x!**

  (note: **fact(0)=1** and **fact(n) = n * fact(n-1)**)