Introduction to Programming in Python

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

More on functions

Docstrings

Python allows inline documentation via *docstrings*. This is just a string that appears directly after the function definition and documents what the function does:

In [ ]:
```
def mypower(x, y):
    """Compute x^y."""
    return x**y
```

- It is customary to use a triple quoted string; these can contain newlines.
- The docstring is shown by the help function

In [ ]:
```
help(mypower)
```

This explains the difference between a comment and a docstring: the former is for the developer, the latter for the user.

Exercise, continued

Add a docstring to the `area` function from last week:

In [ ]:

```
def area(a, b=None):
    if b:
        return a * b
    else:
        return a * a
```

Variable Scope

- Variables defined in functions are local (not visible in the calling scope):

In [ ]:
```
def f():
    z = 1
f()
```

In [ ]:
```
print(z)
```

- The same is true of the input arguments:

In [ ]:
```
def f(num):
    return num**2
```

In [ ]:
```
num
```

Variables defined outside of functions are `global`: they are visible everywhere:

In [ ]:
```
a = 3
def f():
    print(a)
```

In [ ]:
```
f()
```

That is, unless they area "shadowed" by a local variable:

In [ ]:
```
a = 3
def f():
    a = 2
    print(a)
```

In [ ]:
```
f()
print(a)
```

The `global` statement

If we do actually want to act upon the global variable, then we need to be explicit about it:

In [ ]:
```
a = 3
def f():
    global a
    a = 2
    print(a)
```

In [ ]:
```
f()
print(a)
```

Quiz

For each of the following, state what gets printed.

1.

```
def f():
    name = "Alexander"
name = "Simon"
f()
print(name)
```

2.

```
def f():
    global name
    name = "Alexander"
name = "Simon"
f()
print(name)
```

3.

```
def f():
    global name
    name = "Alexander"
name = "Simon"
print(name)
```

4.

```
def f(x):
    x = x + 2
x = 7
f(x)
print(x)
```

5.

```
def f(x):
    x[0] = x[0] + 2
    return x
y = [7]
f(y)
print(y[0])
```

Mutating functions

- That last example was a bit of a curveball.
- Turns out that if you pass a mutable argument (like a `list`) into a function, then changes to that variable are visible to the caller (i.e., outside the function):

In [ ]:
```
def f(y):
    y[0] = 2
```

In [ ]:
```
x = [1]
f(x)
print(x)
```

Splatting and Slurping

- Splatting: passing the elements of a sequence into a function as positional arguments, one by one.

In [ ]:

```
def mypower(x, y):
    return x**y
args = [2, 3]  # a list or a tuple
mypower(*args)  # splat (unpack) args into mypower as positional arguments.
```

- Slurping allows us to create *vararg* functions: functions that can be called with any number of positional and/or keyword arguments.

In [ ]:
```
def myfunc(*myargs):
    for i in range(len(myargs)):
        print("Argument number " + str(i+1) + " was " + str(myargs[i]) + ".")
```

In [ ]:
```
myfunc(3, 5)
```

- I.e., The asterisk means "collect all (remaining) positional arguments into the tuple `myargs`".
- This is essentially how the built-in `print` function works.

Modules

- Python's functionality is organized in *modules*.
- Some of these are part of Python's *standard library* (e.g., `math`). Others are part of *packages*, many of which come preinstalled with Anaconda (e.g., `numpy`).
- Modules need to be imported in order to make them available:

In [ ]:
```
import math
math.factorial(7)
```

- You can use *tab completion* to discover which functions are defined by `math`: after importing, enter `math.` and press the `Tab` key. Alternatively, use dir(math):

In [ ]:
```
print(', '.join(filter(lambda m: not m.startswith("_"), dir(math))))  # just so the output fits o
```

- Note that importing the module does not bring the functions into the *global namespace*: they need to be called as `module.function()`.
- It is possible to bring a function into the global namespace; for this, use

In [ ]:
```
from math import factorial
factorial(7)
```

- It is even possible to import all functions from a module into the global namespace using `from math import *`, but this is frowned upon; it pollutes the namespace, which may lead to name collisions.
- *Packages* can contain several modules. They are imported the same way:

In [ ]:
```
import numpy
numpy.random.rand()
```

- Optionally, you can specify a shorthand name for the imported package/module:

In [ ]:
```
import numpy as np
np.sqrt(2.0)  # note that this is not the same function as math.sqrt
```

- Conventions have evolved for the shorthands of some packages (e.g., `np` for `numpy`). Following them improves code readability.
- For the same reason, it is good practice to put your `import` statements at the beginning of your document (which I didn't do here).

Exercise

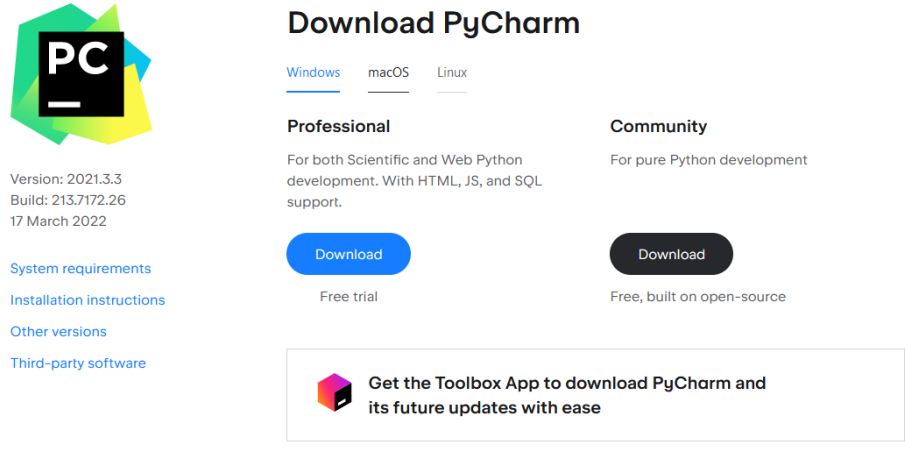
Write a function that takes the radius of a circle as input and returns the area of the circle, using the constant `pi` from the `math` module. Note: the import statement should be at the top of your code, not inside the function.

In [ ]:

PyCharm

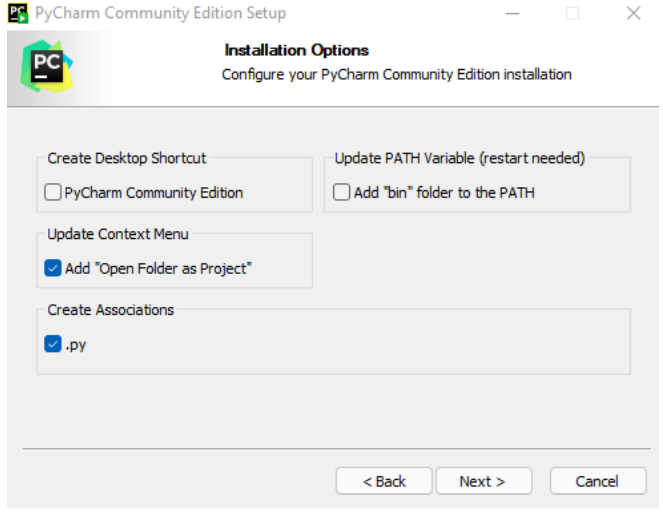
A Python module is essentially just a file with functions and variables defined inside of it. Before we get there, we have to install PyCharm. Let's do this together now.

Download the community edition of PyCharm

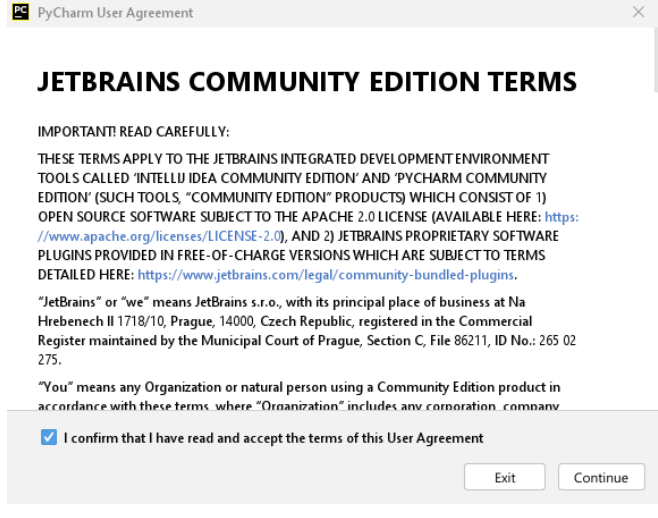


Install it

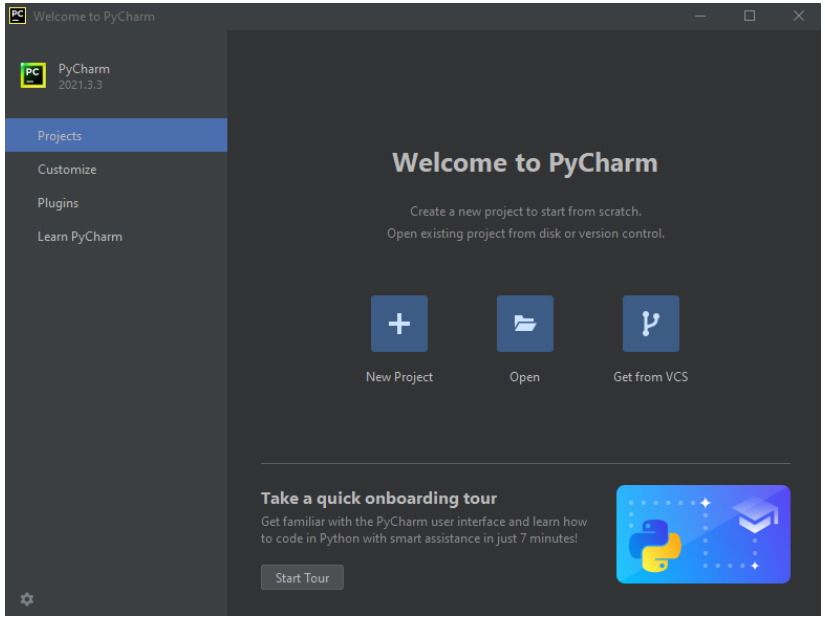
Click next until this screen appears, and check the boxes as shown. Then keep clicking next



Accept the license conditions

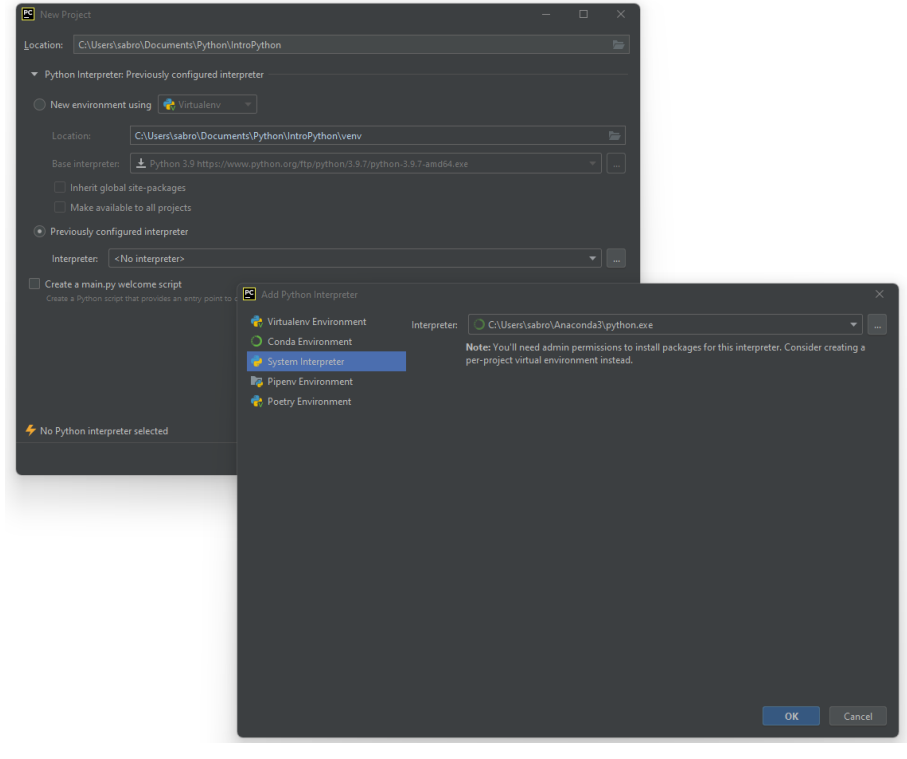


Run PyCharm, click on ”New project”



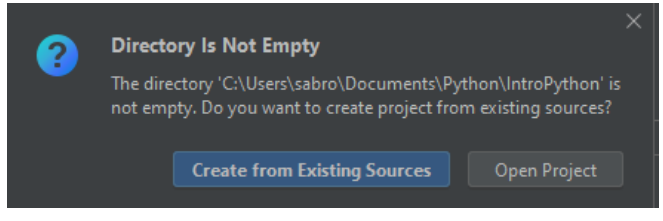
Specify details of new project

- "Location" should point to wherever you keep the course material.
- Choose "Existing interpreter", then choose the system interpreter.



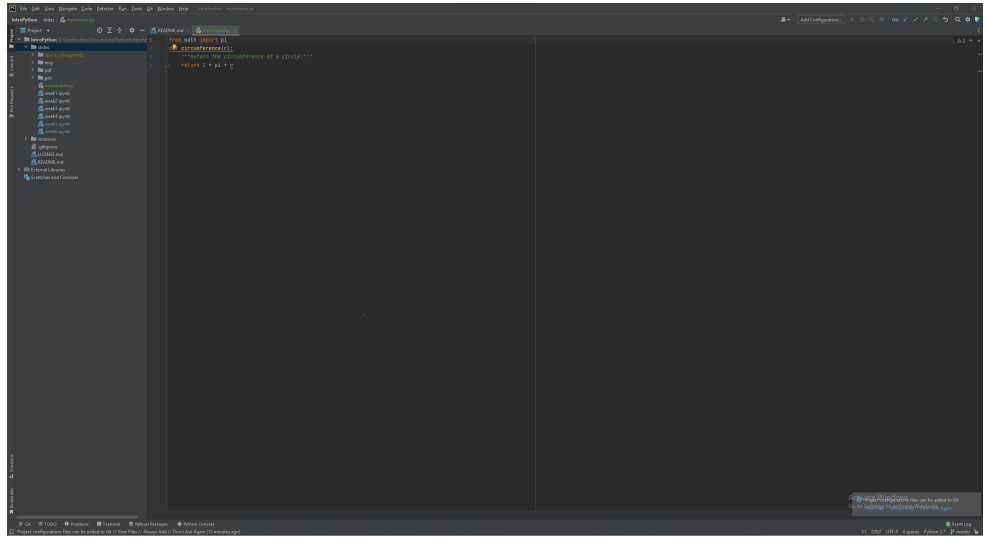
Import files

Choose "Create from existing sources"



Open file

Open the file "mymodule.py" in the "slides" directory



Writing your own modules

As mentioned, modules are just Python files (text files with extension `.py`. Since `mymodule.py` lives in the same folder, we can just import things from it.

In [ ]:
```
from mymodule import circumference
circumference(3)
```

Recap / further reading (optional)

Functions

- https://www.w3schools.com/python/python_functions.asp
- https://python-course.eu/python-tutorial/functions.php

Modules

- https://www.w3schools.com/python/python_modules.asp
- https://python-course.eu/python-tutorial/modules-and-modular-programming.php

Homework

- Exercises 6, 7, 8 from https://www.w3resource.com/python-exercises/modules/index.php
- Create your own Python module containing the function `area` from before. Then, create a notebook that imports the function from the module, and uses it to compute the area of a circle with radius 5.