

Functions Part 1



Admin



- Project questions?
- Still No quizzes in class till just before Thanksgiving but!!
 - In comp143 quiz using computer next week Nov 17-21
- Other concerns?
- Read chapter 8 in the book for this set

Of Code size and readability



- When we first started – just a little python
 - Project1 was just 4ish lines
- Those last programs – dictionaries etc were getting longer
 - Maybe harder to read
 - And still small compared to professional software
- We need a way of organizing our code
 - And preventing the need to copy and paste.

function



- A **function** in programming is a named sub-program that we can call from anywhere in the program (so long as we have imported it for python)
 - Code block that we have named
- In python
 - **def** <function name>(<optional arguments>):
 - Code block
- Example
 - `def say_hello():`
 - `print("hello comp151")`
 - `print("we are learning the 'interesting' stuff now")`

The 'main' function



- The first function that is called typically called main
 - Other languages require it
 - Python just convention
 - The def line for main is against the left edge of the window
 - No indents
 - The code block for main is indented one level
 - When we call the main function it is against the left edge again
- `def main():`
 - `print("hello comp151")`
 - `print("we are learning the 'interesting' stuff now")`
 -
 - `main()`

What should be a function?



- When we decide how code should be broken down into functions
 - Each function should do one thing
 - What is one thing?
 - Print a menu
 - Get data from a file
 - Find the largest thing and report it
 - etc

Functions that return a value



- Some functions just do something
 - The main function from two slides ago
 - Or the print function
- Other functions need to return a value
- Like the input function
- How did we use the input function?
- Lucky volunteer?

Functions that return a value



- To write a function that returns a value
 - Use the return keyword in your function
 - Whatever is to the right of the return is handed back to where the function was called.
- ```
def get_name():
 – return “John”
 –
 –
```
- ```
def main():  
    – my_name = get_name()
```


Lets start



- Let's rebuild our program to work with the game data to use functions
- We'll do three to start.
- load_data
- main
- print_menu

What is going on



- Let's work through what happens when a function is called
 - Draw on board.
 - Maybe use python tutor
 - Linked on resources page
 - To grab this data yourself get it from the resources page
- Use this list of dictionaries for python tutor
- ```
[{'name': 'Terraria', 'release': 2011, 'price': 9.99, 'total_sales': 180000000}, {'name': 'Stardew Valley', 'release': 2016, 'price': 14.99, 'total_sales': 150000000}, {'name': 'RimWorld', 'release': 2018, 'price': 34.99, 'total_sales': 95000000}, {'name': 'Factorio', 'release': 2020, 'price': 35.0, 'total_sales': 91000000}, {'name': "Don't Starve Together", 'release': 2016, 'price': 14.99, 'total_sales': 82000000}, {'name': 'Hollow Knight', 'release': 2017, 'price': 14.99, 'total_sales': 82000000}, {'name': '鬼谷八荒 Tale of Immortal', 'release': 2023, 'price': 19.99, 'total_sales': 69000000}, {'name': 'Project Zomboid', 'release': 2013, 'price': 19.99, 'total_sales': 69000000}, {'name': 'The Binding of Isaac:Rebirth', 'release': 2014, 'price': 14.99, 'total_sales': 66000000}, {'name': 'Bloons TD 6', 'release': 2018, 'price': 13.99, 'total_sales': 66000000}]
```

# Passing information to functions



- Sometimes we want a function to work differently from one use to another
- We want the say\_hello function to greet someone by their name.
- So say\_hello will change what it says every time it runs.
- 

```
import sounddevice as sd
from kokoro_onnx import Kokoro

kokoro = Kokoro("kokoro-v1.0.onnx", "voices-v1.0.bin")

def say_hello(name):
 greeting, sample_rate = kokoro.create(f"Hello {name}, Nice to
 meet you",

 voice="af_sarah", speed=1.0, lang="en-us")
 sd.play(greeting, sample_rate)
 sd.wait()

def main():
 your_name = input("What is your name")
 say_hello(your_name)

main()
```

# Passing information to functions



- Pass information to functions using **arguments** (also known as **parameters**)
  - What goes into the () of a function
- In your book's terms
  - **Parameters** are in the function definition
  - **Arguments** are sent to the the function when you call it.

```
import sounddevice as sd
from kokoro_onnx import Kokoro
```

```
kokoro = Kokoro("kokoro-v1.0.onnx", "voices-v1.0.bin")
```

```
def say_hello(name):
 greeting, sample_rate = kokoro.create(f"Hello {name}, Nice
to meet you", voice="af_sarah", speed=1.0, lang="en-us")
 sd.play(greeting, sample_rate)
 sd.wait()
```

```
def main():
 your_name = input("What is your name")
 say_hello(your_name)
```

```
main()
```

# Passing information to functions



- Let's draw out what happens when we run this code

- Function call diagrams on board
- python tutor doesn't work with external libraries so we can't use it here

- ```
import sounddevice as sd
from kokoro_onnx import Kokoro

kokoro = Kokoro("kokoro-v1.0.onnx", "voices-
v1.0.bin")

def say_hello(name):
    greeting, sample_rate = kokoro.create(f"Hello
{name}, Nice to meet you",
voice="af_sarah", speed=1.0, lang="en-us")
    sd.play(greeting, sample_rate)
    sd.wait()

def main():
    your_name = input("What is your name")
    say_hello(your_name)

main()
```

Passing multiple parameters to functions



- When we have a function with more than one parameter
 - Then we have to pass that same number of arguments
 - By default uses “positional” arguments,
 - First argument is put into first parameter and second argument into second parameter and so on.

```
import sounddevice as sd
from kokoro_onnx import Kokoro

kokoro = Kokoro("kokoro-v1.0.onnx", "voices-v1.0.bin")

def say_greeting(greeting, name):
    speech, sample_rate = kokoro.create(f" {greeting} {name} ",
    voice="af_sarah", speed=1.0, lang="en-us")
    sd.play(speech, sample_rate)
    sd.wait()

def main():
    your_name = input("What is your name")
    say_greeting("How's it going", your_name)

main()
```

Multiple Parameters previously seen



- We have previously seen functions with multiple parameters
- For example the
 - draw_line in dearpygui
 - First is the start point
 - Second is the end point
 - Later specified params (more in a moment)

```
gui_graphics.draw_line((20,20), (200,200),  
color=comp151Colors.SEA_GREEN, thickness=4)
```

Default values for parameters



- When you write a function, you can give a parameter a default value
 - In the function definition type
 - parameter=default value
 - If you think there is a likely value
 - If the user passed an argument, that argument will be used,
 - Otherwise the default value will be used
 - See example

```
def register_class(course_number, course_prefix='Comp'):  
    print(f"Let's register for {course_prefix}{course_number}")
```

```
def main():  
    register_class(152)  
    register_class(161, "Math")
```

main()

Output:

Let's register for Comp152

Let's register for Math161

Default Parameters we've worked with



- Any time I've referred to an 'optional' argument to pass, it actually had a default parameter.
- Remember `draw_arrow`
 -
 - So if you don't pass all 6 arguments, then `line_width` will be 1.
- `draw_arrow` also has default values for several parameters
 - `draw_arrow(p1, p2, ... thickness= 1)`
 - So it turns out that the line thickness is actually optional because it has a default value
 - There are also a few other parameters that we have never sent arguments to (left here as ...)

Keyword arguments



- By default, how do argument values get copied into parameters in python
 - From a few slide back

Keyword arguments



- By default, how do argument values get copied into parameters in python
 - From a few slide back
 - By position. First thing you pass when you call a function becomes value of first parameter etc.
 - Show on board or in pycharm
- But python supports another way

Keyword arguments II



- Python supports **keyword arguments**.

- When you send an argument to a function,
- can use the parameter_name=value, to assign that value to parameter
- regardless of order.
- This is what we've used for most of the optional params – especially in the dearpygui work

```
def register_class(course_number, course_prefix='Comp'):  
    print(f"Let's register for {course_prefix}{course_number}")
```

```
def main():  
    → register_class(course_prefix='MGMT', course_number=152)  
    register_class(161, "Math")
```

Argument passing



- You can technically mix positional arguments and keyword arguments.
 - Usually don't – till you are comfortable with all of this
 - Though dearpygui requires it.
 - Positional arguments have to be first – then keyword
 - Except when you are dealing with oddities of default arguments
 - Eg

```
dpg.draw_line((20,20), (200,200), color=comp151Colors.SEA_GREEN, thickness=4)
```

Mutable vs Immutable Arguments



- When passing an argument to a parameter the location of value of the argument is copied into the parameter
 - Show on board
- For **immutable** objects this is fine – no changes that happen in the Function are seen when the function ends
 - But see following slide

Mutable vs Immutable Arguments



- Let's work through this one
- First with pythontutor then more if needed.

```
def param_mess(number, list):  
    number = number+3  
    list.append(number)  
    return number*number
```

```
def main():  
    demo1 = [2, 3, 5]  
    demo2 = 3  
    result = param_mess(number=demo2, list=demo1)  
    print(f"demo2: {demo2}")  
    print(f"result {result}")  
    print(f"demo1 {demo1}")
```

```
main()
```

Reading



- Finish reading chapter 8
 - This is dense stuff, read it carefully
 - PALs will be covering this too
 - but not during PAL quiz week next week.