In [3]:
```
# Functions: reusable pieces of code
def thing():
    print('Hello')
    print('Fun')
thing()
print('Zip')
thing()
```

```
Hello
Fun
Zip
Hello
Fun
```

In [5]:
```
# Built-in functions that are provided as part of Python -
# print()
# input()
# type()
# float()
# int()
```

In [7]:
```
# Max Function
>>> big = max('Hello world')
>>> print(big)
# Min Function
>>> tiny = min('Hello world')
>>> print(tiny)
```

```
w
```

In [21]:
```
# Type Conversions
# When you put an integer and floating point in an expression, the integer is impli
# You can control this with the built-in functions int() and float()
>>> print(float(99) / 100)

>>> i = 42
>>> type(i)

>>> f = float(i)
>>> print(f)
>>> type(f)

>>> print(1 + 2 * float(3) / 4 - 5)
```

```
0.99
42.0
-2.5
```

In [25]:
```
# String Conversions
# You can also use int() and float() to convert between strings and integers
# You will get an error if the string does not contain numeric characters
>>> sval = '123'
>>> type(sval)
```

```
>>> print(sval + 1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[25], line 4
      1 sval = '123'
      2 type(sval)
----> 4 print(sval + 1)

TypeError: can only concatenate str (not "int") to str
```

In [27]:
```
>>> ival = int(sval)
>>> type(ival)
```

Out[27]:
```
int
```

In [29]:
```
>>> print(ival + 1)
```

```
124
```

In [31]:
```
>>> nsv = 'hello bob'
>>> niv = int(nsv)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[31], line 2
      1 nsv = 'hello bob'
----> 2 niv = int(nsv)

ValueError: invalid literal for int() with base 10: 'hello bob'
```

In [35]:
```
# Building our Own Functions
# • We create a new function using the def keyword followed by optional parameters
# • We indent the body of the function
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
```

In [39]:
```
x = 5
print('Hello')
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
print('Yo')
x = x + 2
print(x)
```

```
Hello
Yo
7
```

In [41]:
```
# Once we have defined a function, we can call (or invoke) it as many times as we l
# • This is the store and reuse pattern

x = 5
```

```
print('Hello')
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
print('Yo')
print_lyrics()
x = x + 2
print(x)
```

```
Hello
Yo
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
7
```

In [43]:
```
# An argument is a value we pass into the function as its inputwhen we call the fun
# We use arguments so we can direct the function to do different kinds of work when
big = max('Hello world')
```

In [45]:
```
# A parameter is a variable which we use in the function definition.
# It is a “handle” that allows the code in the function to access the arguments for

def greet(lang):
    if lang == 'es':
        print('Hola')
    elif lang == 'fr':
        print('Bonjour')
    else:
        print('Hello')

greet('en')

greet('es')

greet('fr')
```

```
Hello
Hola
Bonjour
```

In [51]:
```
# Return Values
# Often a function will take its arguments, do some computation, and return a value

def greet():
    return "Hello"

print(greet(), "Glenn")
print(greet(), "Sally")
```

```
Hello Glenn
Hello Sally
```

In [53]:
```
# • A “fruitful” function is one that produces a result (or return value)
# • The return statement ends the function execution and “sends back” the result of

def greet(lang):
    if lang == 'es':
```

```
        return 'Hola'
    elif lang == 'fr':
        return 'Bonjour'
    else:
        return 'Hello'

print(greet('en'),'Glenn')

print(greet('es'),'Sally')

print(greet('fr'),'Michael')

# When a function does not return a value, we call it a "void" function
# • Functions that return values are "fruitful" functions
# • Void functions are "not fruitful"
```

```
Hello Glenn
Hola Sally
Bonjour Michael
```

In [55]:
```
# Multiple Parameters / Arguments
# • We can define more than one parameter in the functiondefinition
# • We simply add more argumentswhen we call the function
# • We match the number and order of arguments and parameters

def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print(x)
```

```
8
```

In [57]:
```
# To function or not to function...
# • Organize your code into "paragraphs" - capture a complete thought and "name it"
# • Don't repeat yourself - make it work once and then reuse it
# • If something gets too long or complex, break it up into logical chunks and put
# • Make a library of common stuff that you do over and over -perhaps share this wi
```

In [61]:
```
# Rewrite your pay computation with time-and-a-half for overtime and create a funct

def compute_pay(hours, rate):
    overtime_rate = 1.5  # Time-and-a-half rate
    overtime_threshold = 40  # Overtime threshold in hours

    # Calculate regular pay and overtime pay
    if hours <= overtime_threshold:
        return hours * rate
    else:
        regular_hours = overtime_threshold
        overtime_hours = hours - overtime_threshold
        return (regular_hours * rate) + (overtime_hours * rate * overtime_rate)


def get_numeric_input(prompt):
```

```
    while True:
        try:
            return float(input(prompt))
        except ValueError:
            print("Error, please enter numeric input.")


def main():
    # Get hours and rate from user
    hours = get_numeric_input("Enter Hours: ")
    rate = get_numeric_input("Enter Rate: ")

    # Calculate and print pay
    pay = compute_pay(hours, rate)
    print("Pay: ", pay)


if __name__ == "__main__":
    main()
```

```
Pay:  475.0
```

In [ ]: