Starting with python

In [1]:

```
print(5/8)
print(7/10)
```

```
0.625
0.7
```

Python is a pretty versatile language

- quick calculations.
- database-driven
- clean and analyze the results

In [2]:

```
# Division
print(5/8)
# Addition
print(7+10)
```

```
0.625
17
```

In [3]:

```
# Addition, subtraction
print(5 + 5)
print(5 - 5)
```

```
10
0
```

In [4]:

```
# Multiplication, division, modulo, and exponentiation
print(3 * 5)
print(10 / 2)
print(18 % 7)
print(4 ** 2)
```

```
15
5.0
4
16
```

Suppose you have $100, which you can invest with a 10% return each year. After one year, it's 100×1.1=110 dollars, and after two years it's 100×1.1×1.1=121. Add code on the right to calculate how much money you end up with after 7 years.

In [5]:

```
# How much is your $100 worth after 7 years?
print(100*(1.1**7))
```

```
194.87171000000012
```

Variables and types

In [6]:

```
# Create a variable savings
savings = 100

# Print out savings
print(savings)

# Create a variable growth_multiplier
growth_multiplier = 1.1

# Calculate result
result = savings * growth_multiplier ** 7

# Print out result
print(result)
```

```
100
194.87171000000012
```

In [7]:

```
# Create a variable desc
desc = "compound interest"

# Create a variable profitable
profitable = True

#type of variables
a = 0.5
b = "Hello"
c= False
```

In [8]:

```
type(a)
```

Out[8]:

```
float
```

In [9]:

```
type(b)
```

Out[9]:

```
str
```

In [10]:

```
type(c)
```

Out[10]:

```
bool
```

In [11]:

```
savings = 100
growth_multiplier = 1.1
desc = "compound interest"
# Assign product of growth_multiplier and savings to year1
year1 = savings * growth_multiplier
# Print the type of year1
print(type(year1))
```

```
<class 'float'>
```

In [12]:

```
# Assign sum of desc and desc to doubledesc
doubledesc = desc + desc
# Print out doubledesc
print(doubledesc)
```

```
compound interestcompound interest
```

In [13]:

```
#Type Conversion
```

In [14]:

```
# Definition of savings and result
savings = 100
result = 100 * 1.10 ** 7

print("I started with $" + str(savings) + " and now have $" + str(result) + ". Awesom
e!")

# Definition of pi_string
pi_string = "3.1415926"

# Convert pi_string into float: pi_float
pi_float = float(pi_string)
```

```
I started with $100 and now have $194.87171000000012. Awesome!
```

Python Lists

In [15]:

```
# area variables (in square meters)
hall = 11.25
kit = 18.0
liv = 20.0
bed = 10.75
bath = 9.50
```

In [16]:

```
# Create list areas
areas = [hall, kit,liv,bed,bath]

# Print areas
print(areas)
```

```
[11.25, 18.0, 20.0, 10.75, 9.5]
```

In [17]:

```
# Adapt list areas
areas = ["hallway", hall,"kitchen", kit, "living room", liv,"bedroom", bed, "bathroom",
bath]

# Print areas
print(areas)
```

```
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0, 'bedroom', 10.75,
'bathroom', 9.5]
```

Can you tell which ones of the following lines of Python code are valid ways to build a list?

A. [1, 3, 4, 2] B. [[1, 2, 3], [4, 5, 7]] C. [1 + 2, "a" * 5, 3]

List of Lists

In [18]:

```
# house information as list of lists
house = [["hallway", hall],
         ["kitchen", kit],
         ["living room", liv],
         ["bedroom",bed],
         ["bathroom",bath]]
```

In [19]:

```
print(house)
```

```
[['hallway', 11.25], ['kitchen', 18.0], ['living room', 20.0], ['bedroom',
10.75], ['bathroom', 9.5]]
```

Subsetting Lists

In [20]:

```
# Print out second element from areas
print(areas[1])

# Print out last element from areas
print(areas[-1])

# Print out the area of the living room
print(areas[5])
```

```
11.25
9.5
20.0
```

Using a combination of list subsetting and variable assignment, create a new variable, eat_sleep_area, that contains the sum of the area of the kitchen and the area of the bedroom.

In [21]:

```
# Sum of kitchen and bedroom area: eat_sleep_area
eat_sleep_area = areas[3] + areas[7]

# Print the variable eat_sleep_area
print(eat_sleep_area)

# Use slicing to create downstairs
downstairs = areas[0:6]

# Alternative slicing to create downstairs
downstairs = areas[:6]

# Use slicing to create upstairs
upstairs = areas[6:10]

# Alternative slicing to create upstairs
upstairs = areas[6:]

# Print out downstairs and upstairs
print(downstairs)
print(upstairs)
```

```
28.75
['hallway', 11.25, 'kitchen', 18.0, 'living room', 20.0]
['bedroom', 10.75, 'bathroom', 9.5]
```

Q. What will house[-1][1] return?

- A float: the kitchen area
- A string: "kitchen"
- A float: the bathroom area
- A string: "bathroom"

In [22]:

```
print(house[-1][1])
```

```
9.5
```

List Manipulation

In [23]:

```
# Update the bathroom area
areas[-1] = 10.50
# Change "living room" to "chill zone"
areas[4] = "chill zone"

#Extending lists
# Add poolhouse data to areas, new list is areas_1
areas_1 = areas + ["poolhouse",24.5]

# Add garage data to areas_1, new list is areas_2
areas_2 = areas_1 + ["garage",15.45]
```

In [24]:

```
#you can also remove elements from your list. You can do this with the del statement:
x = ["a", "b", "c", "d"]
del(x[1])

#Pay attention here: as soon as you remove an element from a list,
#the indexes of the elements that come after the deleted element all change!
print(x)
```

```
['a', 'c', 'd']
```

In [25]:

```
#remove poolhouse
del(areas[-4:-2])

print(areas)
```

```
['hallway', 11.25, 'kitchen', 18.0, 'chill zone', 20.0, 'bathroom', 10.5]
```

In [26]:

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas_copy
areas_copy = list(areas)

# Change areas_copy
areas_copy[0] = 5.0

# Print areas
print(areas)
```

```
[11.25, 18.0, 20.0, 10.75, 9.5]
```

In [27]:

```
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Create areas_copy
areas_copy = areas

# Change areas_copy
areas_copy[0] = 5.0

# Print areas
print(areas)
```

```
[5.0, 18.0, 20.0, 10.75, 9.5]
```

In [ ]:

```

```